ML Project Lifecycle

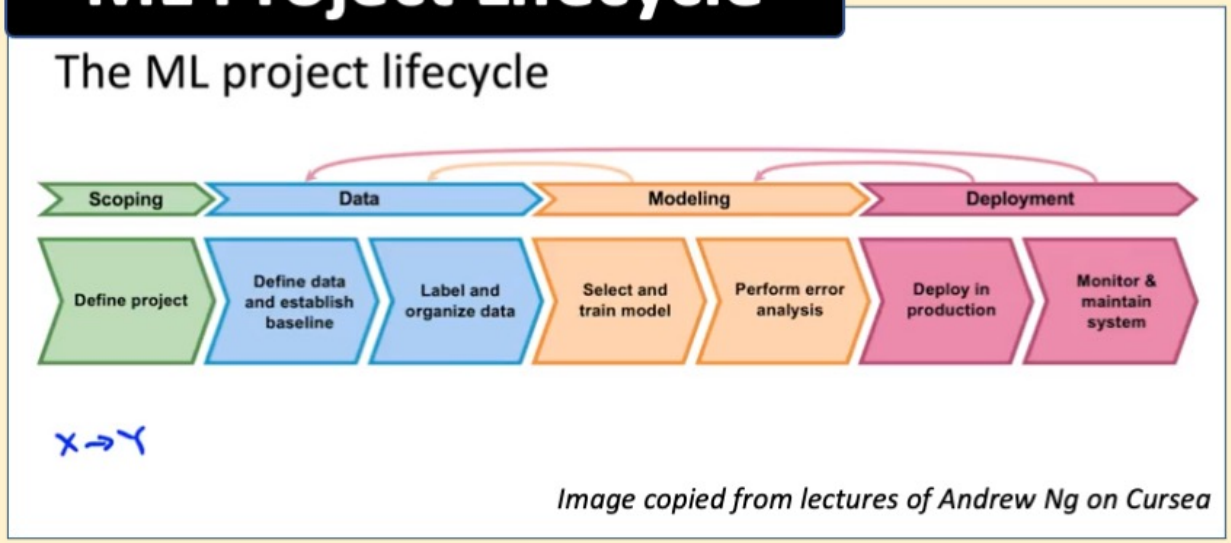


Image copied from lectures of Andrew Ng on Cursea

SOFTWARE DEV. ISSUES
– COMMON QUESTIONS -

Do you need Realtime or Batch predictions
- eg: Batch prediction can be used, for analysing X-ray images from patients over night

Does your prediction system runs on the cloud or on edge device/browser
- eg: factory can not stop working when it looses internet connection
- the same with cars, airplanes etc...
- modern web browsers - ML model can be deployed directly on client machine

How much compute recourse do you have
- CPU/GPU/memory
- do I need to reduce ml model complexity?

Latency, throughput (QPS)
- measured in **QPS (Query per Second) -** ie how many predictions you need to handle per second, given your recourses?
- eg. you may get **latency requirement** with specific time budget assigned to a given task like 500ms. for entire speech translation, from that you may have only300 Ms for making prediction itself
- if you get a requirement that need to handle 1000QPS - you must make sure to test the system to learn whether it can handle that

SCOPING STAGE

define the project:
- what is the business question?
- how I am going to solve that?
- can I solve that problem with AI?
- Do I even need an AI?

decide on key metrics:
- eg: acc, latency, throughput

estimate recourses
& preliminary timeline

DATA STAGE

A. Define the data & establish baseline

Typical data problems:
- How my data are labelled?
- are my data labelled consistently?
- what trimming, standardisation, normalization was/should be used?

B. Label and organize your data
- Use best practices to spot inconsistencies
- Ask people to consistently label the data
- Generally talk with the people who create/collect/label the data

MODELLING STAGE

ML System has three major components
- the code
- hyperparameters
- Data

Academia vs Product dev. Team approaches
- academics often use fixed dataset and manipulate with hyperparameters and the code
- dev. team
  - often fix the code (like with skin diagnostic AI)
  - use different data or data treatments methods
  - plus they test a lot of hyperparameters to tune the models,

SANITY-CHECK
- try to overfit the model to check your code and the model implemented, use the same data for train and test, use only few data points, eg 3 images.
- use small, known datasets for unit test, & to try out if everything works out
- try one training example - it should give back the results,

DEPLOYMENT STAGE

STATISTICAL/ML MODEL ISSUES

DATA DRIFT
- changes in data after model deployment
- Gradual changes
  - eg: language changes over the time
- Sudden shock
  - eg. change in behaviour of people in e-commerce after covid (it started antifraud alarms)

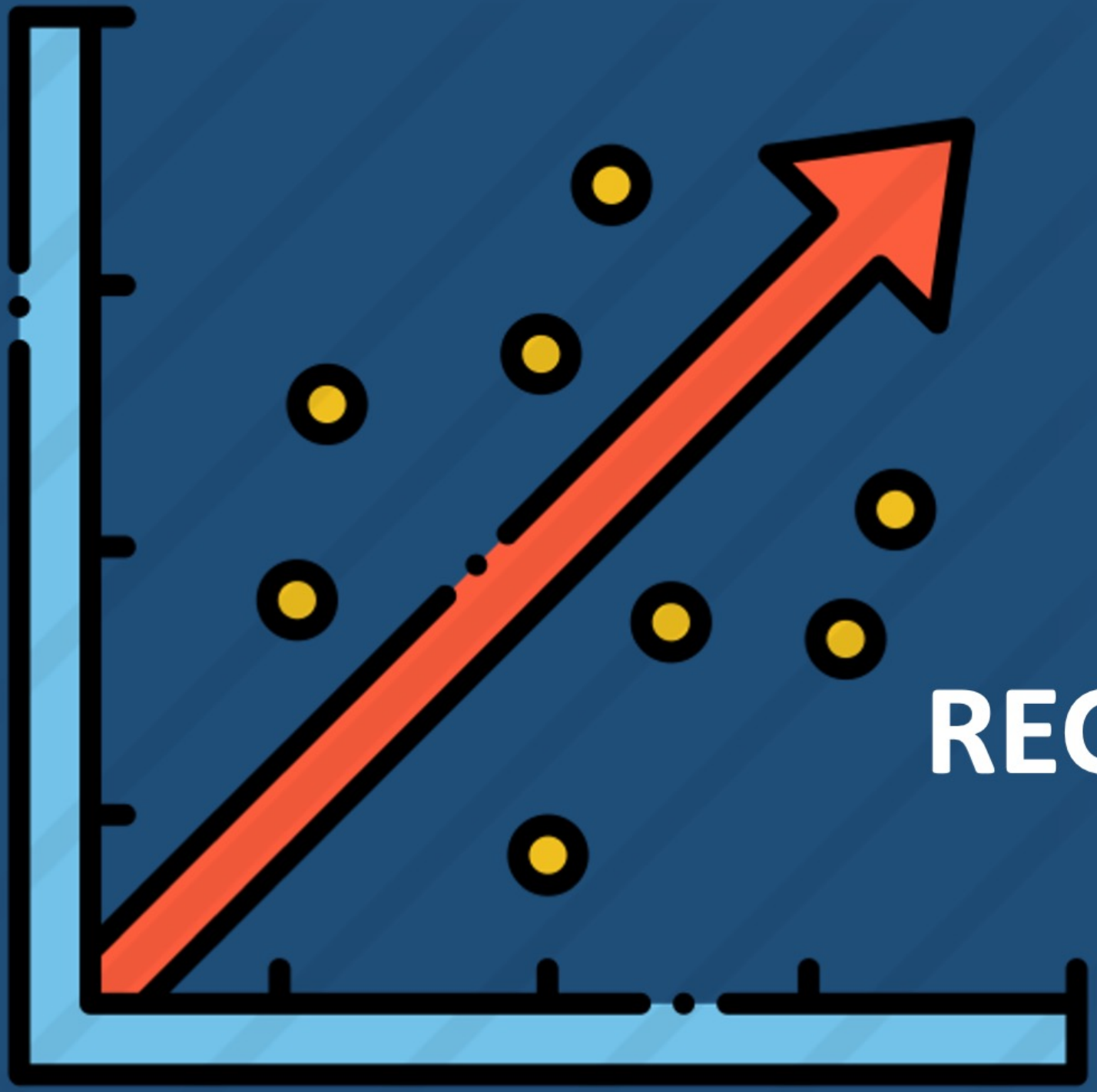
CONCEPT DRIFT
- changes in mapping x -> y with ml model,
- data can be the same but show different pattern,
- eg: the same person, now buys different things,

how fast it can happen?
- **User data**
  - very steady
  - slow changes in behaviour of large groups of people
  - Exceptions (covid-19, new trends, viral videos)
- **Enterprise data**
  - fast changes
  - eg. one CEO decision may change the supplier, and the type of product that we are using or monitoring with ml system

OTHER ISSUES

LOGGING
- ie. what data do you plan to collect from the day to day operations of the model
- logging takes time and recourses
- privacy and data ownership issues

Security & Privacy
- requirements for S&P may be very different for different types of systems
- eg. **clinical data** - high requirements for securing patients records
- always ask how sensitive the data are, and what are regulatory obligations and constraints



REGRESSION ANALYSIS

'image: Flaticon.com'.

by Pawel Rosikiewicz

Regression Analysis & Generalized Linear Models

www.SimpleAI.ch

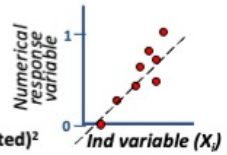
REGRESSION ANALYSIS

Fitting a function f(.) to datapoints yi=f(xi) under some error function. ie a method for estimating relationships between a dependent v. (response/outcome) and ≥1 independent variables, called 'predictors', 'covariates', 'explanatory variables', 'features'

- **simple linear regression;** one x, & one scalar y var.
- **multiple linear regression;** >1x var & one scalar y var.
- **multivariate linear regression;** multiple correlated dependent var's are predicted

LINEAR REGRESSION

Linear regression; fits the line/hyperplane by minimizing RSS, MSE; MAE error function



Model: $y = w0 + w_1x_1 + w_2x_2 + ... + w_nx_{n5}$

Cost Function, eg: MSE = 1/n Σ(y_true – y_predicted)²

OLS – Ordinary Least Squares

Commonly used technique for estimating coefficients (w) of linear regression equations.

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

$\hat{\beta}$ = ordinary least squares estimator
X = matrix regressor variable X
$\top$ = matrix transpose
y = vector of the value of the response variable

Alternatives: eg: grid search or SDG.

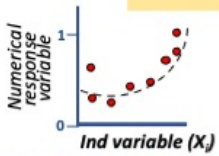
Main advantage; OLS find optimal solution analytically, it uses all train data, at once to solve well-understood equation, that provides one, and only one solution, unlike gradient methods

Requirements:

- *Input data; should be full rank matrix*
- *No ill-conditioning and no collinearity, in the data*
- *Error f. – must be convex – we use MSE* error function wit
- For more , see slides "testing linear regression assumptions"

Weighted Least Squares (WLS): **special form of** OLS used when residuals, are not homoscedastic, eg. when larger target values produce larger errors, - we add weights to each target value, reducing their effect on coefficients.
ALTERNATIVES: transform target variables (log, sqrt, box-cox), or use cost function more resilient to outliers MAE, Hubert loss

POLYNOMIAL REGRESSION



A special case of Multiple linear regression (Linear regression) which estimates the relationship as an nth degree polynomial. to model non-linear x-y relationship

eg: y= w0 + w1x + w2x^2 == a statistical estimation problem is still linear

Degree, or the higher-degree terms; new explanatory var. resulting from the polynomial expansion of the "baseline" var. eg x^2, x^3, x^4
If we say the equaion has the degreee ==3, it is y= ax + bx^2 + cx^3

Polynomial expansion; a process of adding higher degree terms to the model, and creating so called extended/augmented feature vectors.

Polynomial basis; extended feature vector

Interpretation problem; It is often difficult to interpret the individual coeff. in a polynomial regr. fit, since the underlying monomials can be highly correlated (eg. x and x^2 may have r=0.97). It is generally more informative to consider the fitted regression function as a whole. **Point-wise or simultaneous confidence bands** can then be used to provide a sense of the uncertainty in the estimate of the regression function.

ALTERNATIVES; Nonparametric regression, regression trees, SVM. You can also try other types of input transformation, eg sqrt

REGRESSION WITH REGULARIZATION

Regularisation combats over-fitting by reducing model variance and complexity. The main idea, is to add constraint to the amplitude od the coefficients to cost function, called **PENALTY, or penalization term**. It is a Procedure that allows, to keep weights relatively small (L2 regularization), or select most relevant coefficients (L1 r.). to control the regularization strength, we are using **ALFA/LAMBDA** hyperparameter [0,1],

Lasso Regression (L1 regularization)

Linear regression with L1 penalty term added (Can fit either a line, or polynomial minimizing the error each datapoint & the weighted L1 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} |\beta_j|$$

$$f_\beta(x_i) = f_\beta^{poly}(x_i) \; or \; f_\beta^{linear}(x_i)$$

Typically, used for parameter selection,

Ridge Regression (L2 Regularization)

... L2 norm of the function weights

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} \beta_j^2$$

$$f_\beta(x_i) = f_\beta^{poly}(x_i) \; or \; f_\beta^{linear}(x_i)$$

Used for estimating the coeff. of multiple-regr. models in scenarios where independent variables are highly correlated (multicollinearity)

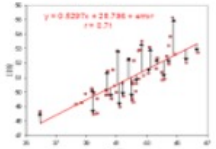
ERROR FUNCTION

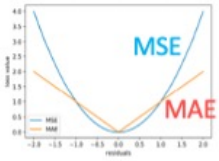
Linear regression may be used with different error functions, that affect what optimalization method we can use, or how much weight is given to potential outliers. Eg: sklearn implements Hubert r., used to dealing with outliers

There are several similar cost functions that are often reported:

- **MSE,** Mean of Squared Errors
- **RMSE/RMSD,** root-mean-square error/deviation
- **RSS,** Residual Sum of Squares

We will get the same set of optimal parameters using RSS, MSE, & RMSE, because the alg. finds the weight combination with the lowest error





➢ **MSE/MAE**

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad \text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.$$

$$MAE = \frac{1}{N} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

➢ **Hubert loss**

$$L(e) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \le \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

- **δ;** det. when the function becomes linear
- **e;** residual e=(y-y-hat)
- **L(e);** loss value for one data point

➢ **Which cost function I should report to interpret the results**

- **RMSE/RMSD, are easier to report, then RSS, because** it has the same units as the target variable, and it is Independent on data point number. $
- **Use RSS/MSE etc...** when baseline is Mean.
- **Use & report MAE:** When baseline is a MEDIAN, Caution, its not possible with all functions. Eg MAE can be used with sub gradient descent approach, that is often implemented in neural networks

GLMs – Genelized Linear Models

Used for modelling response variables that are bounded or discrete, or to model response for var. that have large scale, and or skewed distributions. Linear regression is also a GLM, with identity link function (see below). The term "general" linear model (GLM) usually refers to conventional linear regression models for a continuous response variable given continuous and/or categorical predictors. Caution ***GLM are not General linear Models***

GLM three components

1. **Linear predictor:** a linear function used to target values transformed by the link function, The main idea, is that there is a linear relationship between input data and the target variables transformed with the link function.
2. **Link Function:** a fixed function, (it has no weights that are being optimized) that transforms the target variable. It indicates how the expected value of the response relates to the linear combination of explanatory variables. Examples are:
   - Ln(y) for Ppoisson regression
   - Lod(odds) ie Logit(y) for logistic regression
   - Y'=y, ie identity l.f. for ordinary linear regression
3. **Probability distribution:** specifies the probability distribution of the response variable; e.g., normal distribution for in the classical regression model, or binomial distribution for Y in the binary logistic regression. This is the only random component in the model; there is not a separate error term.

Most Known GLMs

- **POISSON R; for count data.**
- **LOGISTIC & PROBIT R; for binary data**
- **MULTINOMIAL LOGISTIC & PROBIT R; for categorical data.**
- **ORDERED LOGISTIC & PROBIT R.; for ordinal data**

GLM vs Traditional OLS regression

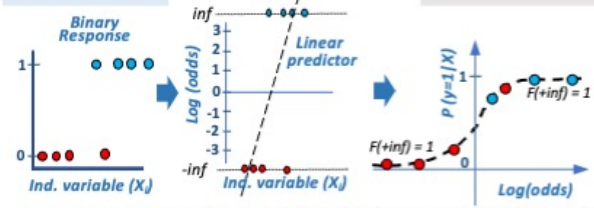
- no need to transform the response to have a normal distribution.
- The choice of link is separate from the choice of distributions, == more flexibility (eg: R function glm, sklearn also , allow selecting all three components separetely)
- The models are fitted via MSE, so likelihood functions and parameter estimates benefit from asymptotic normal and chi-square distributions.
- The inference tools and model checking that used for logistic and Poisson regression apply for other GLMs too; e.g., Wald and Likelihood ratio tests, deviance, residuals, confidence intervals, and overdispersion

Logistic regression

- CLASSYFICATION METHOD -

Logit function ***Ln(p/(1-p))***

Distribution Binomial



Sigmoid function

$$F = \frac{e^{log(odds.)}}{(1 + e^{log(odds)})}$$

Log Regr. computes the probability of a data point being in one class: p(y=1|x)=sigma(fx), where f(x)=xw, is a linear regression function f(x)=xw, and sigma ∈ [0,1]. Y is a binary response variable ∈{0,1}.

Weights (w) in f(x) are found by minimizing **cross-entropy (CE) loss function**, using **max. likelihood estimation alg. (MLE).** MLE estimates params of a likelihood function, given the observed data, to the point called maximum likelihood estimate.

Why Sigma function?; cdf of logistic distrib. has a higher kurtosis than normal .distrib. (wider tails), and cdf S shape is more flat. Thus it is used more in natural sciences

LOGREG vs PROBIT R.; probit r. uses cdf of norm. distrib. Applied more in social sc. and economy, (heteroscedastic data)

Muticlass classification; 3 approaches available

a) Softmax Regr;
b) b) OvO; one vs One,
c) c) OvR; one vs rest

Solvers in sklearn ; {liblinear, lbgf, SAG and SAGA},

- SAG & SAGA with large datasets,
- (L1, L2), use on scaled data

by Pawel Rosikiewicz

GLMs; the three most known examples

www.SimpleAI.ch

Linear regression

Notation

$y_i = x_i^T w$

$y_i \sim N(mu, sd)$

Assumptions:

- ***Validity of Linear Models (linear relationship between X and y)***
- ***No multicollinearity in the data***
- ***Normally distributed residuals***
- ***Homoscedastic residuals / constant variance***
- ***No auto-correlation / independence of errors***

Cost Function

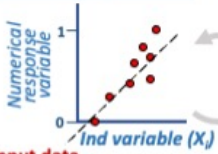
MSE
MAE
Hubert loss

Solvers

LS
Gradient methods
MLE

Identity function $y_i = y_i$

Distributions: *Normal Distr.*



In basic form, no transformation of target variable is done

We may transform target variable in case of residuals heteroscedascity (see further)

Input data

- *Continuous, Discrete,*
- *Categorical, Ratio, Binary*

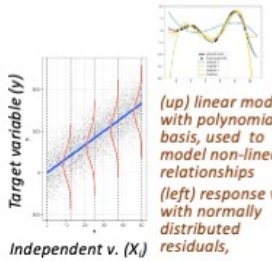
Target Variable

- *Continuous, numerical variable, range –inf - +inf*

Output data

- *Continuous numeric variable, from –inf to +inf*

It is possible to restrict results only to positive values (see sklearn documentation)



(up) linear model with polynomial basis, used to model non-linear relationships

(left) response var. with normally distributed residuals,

➢ COMMONLY USED MODELS

OLS - Ordinary Least Squares

MSE cost f. + LS

- *Input data; full rank matrix*
- *No ill-conditioning*
- *Error f. – must be convex*

WLS - Weighted Least Squares

*MSE *weights cost f. + LS*

- *used when there is a considerable amount ot heterocedascity in residuals.*
- *No autocorrelation !*
- *Weights added, by uset to est.fit()*

GLS - Generalized Least Squares

*MSE *weights cost f. + LS*

- *For problems with autocorrelation and heteroskedasticity*
- *Caution: it creates a corr. Matrix n^2*
- *Implemented in **statmodels (python)***

➢ COMMON ISSUES

OVERFITTING

- Regularization,
- get more data,
- Simplify the model

UNDER FITTING

- Polynomial expansion,
- Feature engineering & Scaling
- F. selection

HETEROSCEDASCITY

- Target variable transformation: log, sqrt or box-cox (see slides on sklearn tranformers)
- WLS, or other type of weights

OUTLIERS

- Identify with Z-score, Cooks distance,
- Huber regression
- Quantile regression (sklearn)

➢ MODEL EVALUATION

- Trian/test errors,
- R^2 (problems with R^2 expansion)
- p values
- Estimate Model complexity (eg. weight values, smaller are better)
- Test model assumptions, and perform analysis of residuals

sklearn.linear_model

> ***LinearRegression()**, OLS, and WLS*
> ***SDGRegressor()**, SDG implementation,*
> ***HuberRegressor()**; best for "small data"*
> ***Ridge()**; OLS + L2 penalty*
> ***ElasticNet(); Lasso()***
> ***BayesianRidge()**; used as OLS, but more reliable for illconditioned data*
> *Many functions have CV version for cross validation eg: **RidgeCV, LassoCV***

Logistic regression

Notation

$logit(p_i) = x_i^T w$

$p_i = e^{log(odds.)} / (1 + e^{log(odds)})$

$y_i \sim Bern(p_i)$

Cost Function

MLE
Croiss-entropy

Assumptions:

- ***Binary target variable***
- ***Independent observations***
- ***Linear relationship between independent variables and target log odds***
- ***No multicollinearity in the data***

Solvers

each solver works only with a subset of penalty terms, and some, like liblinear can only use OvR strategy for multiclass classif.

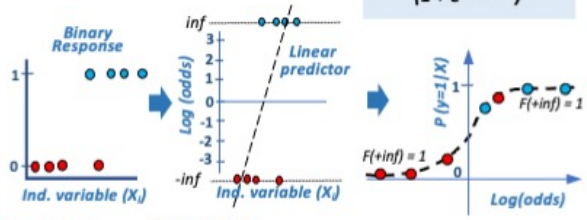
- ***Lbfgs** – for small nr of samples, fast convergence eg maxiter =100*
- ***SAG & SAGA** - for large datasets, Stochastic average gradient, based on GD*
- **newton-cg,** ; for large, sparse data

Logit function $Ln(p/(1-p))$

Distribution Binomial

Sigmoid function

$$F = \frac{e^{log(odds.)}}{(1 + e^{log(odds)})}$$



Input Data

- *Continuous, Discrete,*
- *Categorical, Ratio, Binary*

Target Variable

- *Binary, or multiple class*

Output Data

- the probability of a data point being in the positive class: p(y=1|x)=sigma(fx), where f(x) is a linear regr. fn on input data and logit values, sigma is a binary target variable ∈{0,1}.
- Typically, logreg requires large sample size
- Best if the data have multivariate normal d.
- No strongly influential outliers

➢ IMPORTANT

(1) SCALE THE DATA
(2) REGULARIZATION

Its very important in logreg, especially in high-dimensional data, where rare combinations of features, would drive model to overfitting, and increase coeff. toward inf.

- **Early stopping**
- **L2 regularization**
- **ElasticNet**
- **L1, - sparse data**

➢ COMMON ISSUES

CLASS IMBALANCE

- Target variable transformation: log, sqrt or box-cox
- WLS, or other type of weights

➢ MODEL EVALUATION

- **Trian/test Acc, Recall, Sensitivity,**
- **ROC & PR curves, AUC**
- **R^2 (calculated differently)**
- **p values, AIC, BIC**

sklearn.linear_model

> ***LogisticRegression(); LogisticRegressionCV()***

Alternatives and other implementations

> ***SGDClassifier();** more efficient with big data: Linear & SVM models are imple-mented, including logreg, with SGD & mini-batch training. Has large # of options, eg: many cost f's (eg: L1, L2, hinge loss, hubert_loss, log-logistic regr.), Caution: sensitive to feature scaling.*
> ***Perceptron();** Linear perceptron classifier, hares the same underlying implementation with SGDClassifier (have equivalent form)*
> ***sklearn.svm.LinearSVC();** Linear Support Vector Classification, liblinear implementaiton,, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.*
> ***RidgeClassifier(),** OLS+L2 penalty, may be faster with large number of classes*

➢ MULTILABEL CLASSYF.

- Softmax Regr;
- OvO; one vs One,
- OvR; one vs rest

in multiclass classf, skleanr uses Cross-entropy cost function (CE) and cdf of multinomial cdf

➢ PROBIT REGRESSION

Alternative to logreg. With normal distribution, and integrals used to claulate the P(y=1|X) both models yield similar results, except:

(i) if IIA assumption is violated (ie. no independence between multiple classes).

(ii) the probate incorporates non-linear effects of X as well (its always form 0 to 1)

Poison regression

Notation

$ln(\lambda_i) = x_i^T w$

$y_i \sim Poison(\lambda_i)$

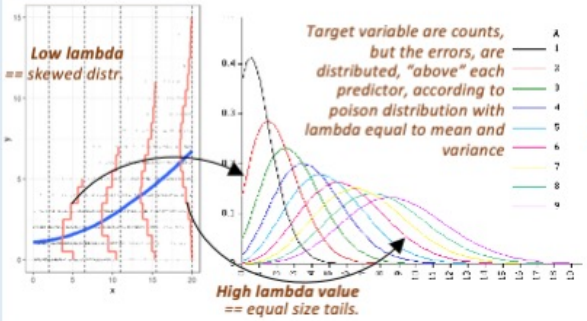
Assumptions:

- ***Poisson Response** The response variable is a count per unit of time or space, described by a Poisson distribution. + these must by positive integers (eg. 0,1,2,3 ...)*
- ***Independence** of observations*
- ***Mean=Variance,** Conts must follow posion distrib with mean and variance = λ*
- ***Linearity** The log of the mean rate, log(λ), must be a linear function of x.*

Poisson regression; a special case of the GLM, where the random component (error distribution in target variable) is specified by the Poisson distribution. Unlike the binomial distribution, which counts the number of successes in a given number of trials, a Poisson count is not bounded above. When all explanatory variables are discrete, the Poisson regression model is equivalent to the **log-linear model**, which.

logarithm *Ln()*

***Model:** Poison regression*
***Distribution:** Poison*



➢ COMMON ISSUES

ZERO-INFLATION

ZIP – zero inflated Poison regression - model used in the present of excess zero values (eg. insurance claims)

https://timeseriesreasoning.com/contents/zero-inflated-poisson-regression-model

sklearn.linear_model

> ***PoissonRegressor();***

Cost Function

Perfect Cost Function

- Should be robust to outliers and easy to optimize
- The functions, that are differentiable at each point, and convex are easy to optimize.
- MSE/RSS are smooth funct., but sensitive to outliers
- MAE is robust, but difficult to optimize, it is not differentiable at 0.
- Huber loss f. combines some advantages of RSS & MAE, but it introduces new hyperparameter δ (epsilon in sklearn)
- You may also use weight, like with WLS approach

Statistical vs computational trade-off

Desired statistical properties are

- Cost function should be robust to **outliers**

Desired computational properties are

Cost function should be

- Smooth & symmetrical
- Convex (easy to find min.)
- have only one global optimum (see 1st and 2nd degr. optimality conditions)

POPULAR ERROR FUNCTION

Linear regression may be used with different error functions, that affect what optimalization method we can use, or how much weight is given to potential outliers. Eg: sklearn implements Hubert r., used to dealing with outliers

➢ MSE / RSS / RMSE

FEATURES

- **Symmetric**
- **Sensitive to outliers**
- **Smooth** around zero,
- **Scale-dependent,** it should not be used to compare error on different datasets,
- Used only to compare forecasting errors of different models **for a particular dataset**

We will get the same set of optimal parameters using RSS, MSE, & RMSE, because the alg. finds the weight combination with the lowest error !

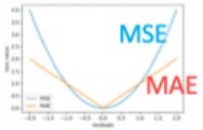
There are several similar cost functions that are often reported:

- **MSE,** Mean of Squared Errors
- **RMSE/RMSD,** root-mean-square error/deviation
- **RSS,** Residual Sum of Squares

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

$$MAE = \frac{1}{N}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$



➢ MAE

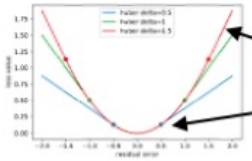
- **Symmetric**
- **Robust to oultiers**
- **Not smooth** around zero, ie. it changes abruptly, & its hard to find optimal parameters,
- **MAE values are easy to INTERPRET**, eg: if MAE=3, it means that scores = ±3 units,

➢ HUBER LOSS

- **Symmetric**
- **Robust to outliers**
- **Smooth around zero,**
- Introduces additional parameters, δ, that is not always corresponding to variable values in the feature speace.

$$L(e) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \le \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

- **δ;** det. when the function becomes linear
- **e;** residual e=(y-y-hat)
- **L(e);** loss value for one data point

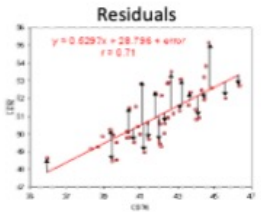


>δ behaves as MAE.

δ< behaves like RSS

➢ Which cost function I should report to interpret the results

- **RMSE/RMSD, are easier to report, then RSS, because** it has the same units as the target variable, and it is Independent on data point number.
- **Use RSS/MSE etc...** when baseline is Mean.
- **Use & report MAE:** When baseline is a MEDIAN, Caution, its not possible with all functions. Eg MAE can be used with sub gradient descent approach, that is often implemented in neural networks



Example of two c
& MAE, plotte
parameter/weigh

mean-tweedie-deviance

gression_non_normal_loss.html#sph
ormal-loss-py

Cost Function

Cost Functions
- Objective f., loss f. -

Perfect Cost Function

- Should be robust to outliers and easy to optimize
- The functions, that are differentiable at each point, and convex are easy to optimize.
- MSE/RSS are smooth funct., but sensitive to outliers
- MAE is robust, but difficult to optimize, it is not differentiable at 0.
- Huber loss f. combines some advantages of RSS & MAE, but it introduces new hyperparameter δ (epsilon in sklearn)
- You may also use weight, like with WLS approach

Statistical vs computational trade-off

Desired statistical properties are

- Cost function should be robust to **outliers**

Desired computational properties are

Cost function should be

- Smooth & symmetrical
- Convex (easy to find min.)
- have only one global optimum (see 1st and 2nd degr. optimality conditions)

It is usually advisable to choose an error function appropriate for the distribution of noise in your target variables (McCullagh and Nelder 1989). Eg, in linear regression you assume, that residuals (ie. noise coming if the perfect model is added) are normally distributed, and that distribution, with its params are basis of the Maximul Likelihood estimator. Eg. in count data, the distribution of errors, is often following poison distribution, hence it is used for optimalization. But if your software does not provide a sufficient variety of error functions, then you may need to transform the target so that the noise distribution conforms to whatever error function you are using.

Cost Functions Examples

➢ RSS - Square of the residuals

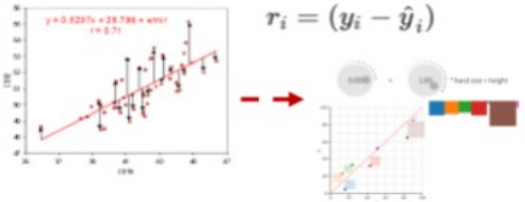
- **Symmetric**
- **Sensitive to outliers**
- **Smooth** around zero,
- **Scale-dependent,** it should not be used to compare error on different datasets,
- Used only to compare forecasting errors of different models **for a particular dataset**

SEVERAL SIMILAR COST FUNCTIONS

- **MSE,** Mean of Squared Errors
- **RMSE/RMSD,** root-mean-square error/deviation
- **RSS,** Residual Sum of Squares
  - **residuals (ri);** difference between approx. (y-hat) & observed y value
  - **np.polyfit()** OLS solution

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2.$$



$$r_i = (y_i - \hat{y}_i)$$

Classification	Regression
Single label: Categorical crossentropy	**Continuous values:** Mean squared error Mean absolute error Mean squared logarithmic error
Multi-label: Binary crossentropy Squared hinge	**Discrete values:** Poisson

https://scikit-learn.org/stable/modules/model_evaluation.html#mean-tweedie-deviance

sklearn.metrics.mean_poisson_deviance

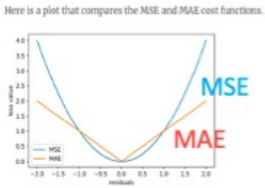
Poisson regression and non-normal loss

https://scikit-learn.org/stable/auto_examples/linear_model/plot_poisson_regression_non_normal_loss.html#sphx-glr-auto-examples-linear-model-plot-poisson-regression-non-normal-loss-py

MAE

- **Symmetric**
- **Robust to oultiers**
- **Not smooth** around zero, ie. it changes abruptly, & its hard to find optimal parameters,
- **MAE values are easy to INTERPRET**, eg: if MAE=3, it means that scores = ±3 units,

Imp
We will g
set of optimal
parameters
MSE, &



$$MAE = \frac{1}{N}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

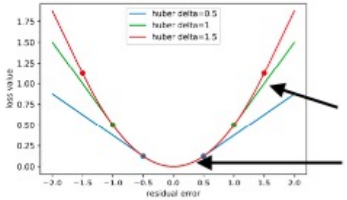
HUBER LOSS

- **Symmetric**
- **Robust to outliers**
- **Smooth around zero,**
- Introduces additional parameters, δ, that is not always corresponding to variable values in the feature speace.

$$L(e) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \le \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

Where:

- **δ;** det. when the function becomes linear
- **e;** residual e=(y-y-hat)
- **L(e);** loss value for one data point



>δ behaves as MAE.

δ< behaves like RSS

```
# Create a linear regression with RSS loss
lr_squared = SGDRegressor(loss='squared_loss', penalty='none', max_iter=1000, tol=1e-3)
```

RSS

```
def RSS(y, y_pred):
    return np.sum(np.square(y - y_pred))
```

RMSE

```
# Root mean squared error
def RMSE(y, y_pred):
    mse = np.mean(np.square(y - y_pred)) # MSE
    return np.sqrt(mse) # RMSE
```

MAE

```
# Mean absolute error
def MAE(y, y_pred):
    return np.mean(np.abs(y - y_pred))
```

by Pawel Rosikiewicz

REGULARIZATION

www.SimpleAI.ch

WHEN TO USE EACH TYPE

- **Lasso** — data have few significant predictors and the magnitudes of the others are close to zero, often used to select the most important features.
- **Ridge** - there are many large predictors of about the same value
  - multicolinearity
- **ElasticNet** —there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.
- **LARS** - alternative to ridge and ElasticNet for multiple colinear features,
  - iterative approach,
  - More effective for p>>n datasets, and alfa exploration

L1 REGULARIZATION (LASSO)

- least absolute shrinkage and selection operator
- Prevents the weights from getting too large (defined by L1 norm).
- L1 reg. introduces sparsity in the weights. It forces more weights to be zero

$$loss = error(y, \hat{y}) + \lambda \sum_j |\beta_j|$$

LASSO do well if there are few significant predictors and the magnitudes of the others are close to zero

LASSO LIMITATIONS

- **If p>>n**, where p is large, and n "small", == LASSO selects at most n variables before it saturates
- **No group selection:** if there is a group of highly correlated variables == LASSO selects only one such var. and ignores the rest

L2 REGULARIZATION (RIDGE)

- Prevents the weights from getting too large (defined by L2 norm).
- Developed as the solution to the imprecision of least square estimators when linear regression models have some multicollinearity (highly correlated features).

$$loss = error(y, \hat{y}) + \lambda \sum_j \beta_j^2$$

works well if there are many predictors of about the same magnitude. This means all predictors have similar power to predict the target value.

L2 LIMITATIONS

- includes all the predictors in the final model (also a pros)
- == unable to perform feature selection.
- Shrinks coeff^s towards zero.

ELASTIC NET

includes L1 and L2 penalty terms with lambda1 &2, or their ratio (sklearn solution). Often considered "the best choice", however it comes on the expense of having one extra parameter.

- unique minimum (strictly convex) - The quadratic penalty term makes the loss function strongly convex
- Implemented. In skleanr for linear regression, logistic regression and linear support vector machines

$$argmin_\beta \sum_i (y_i - \boldsymbol{\beta}' \boldsymbol{x}_i)^2 + \lambda_1 \sum_{k=1}^{\kappa} |\beta_k| + \lambda_2 \sum_{k=1}^{\kappa} \beta_k^2$$

LARS

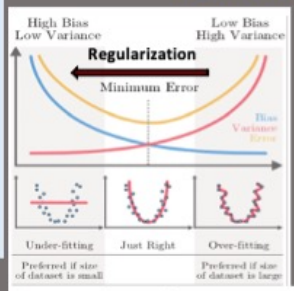
- least-angle regression (LARS)
- for high-dimensional data
- Allows selection of variables and their coefficients to include in the model

Effective for p>>n datasets

If 2 var.s are equally correlated with the response var., it will provides similar weights for both of them

Sensitive to collinearity

HOW REGULARIZATION WORKS?



It combats over-fitting by reducing model variance and complexity

- The main idea, is to add constraint to the amplitude od the coefficients to cost function, called PENALTY, or penalization term. It is a Procedure that allows, to keep weights relatively small, or select most relevant coefficients.
- to control the regularization strength, we are using ALFA/LAMBDA hyperparameter [0,1],

Larger the weights == more complex model == more chances of overfitting

ALFA/LAMBDA

Controls the strength of the penalty term

- controls the amount of attention that the learning process should pay to the penalty == the amount to penalize the model based on the size of the weights.
- ***If the penalty is too strong, the model will underestimate the weights and underfit the problem. If the penalty is too weak, the model will be allowed to overfit the training data.***
- value between 0.0 (no penalty) and 1.0 (full penalty)
  - **0.0 == low bias (high variance),**
  - **1.0 == high bias (low variance).**

HOT TO FIND OPTIMAL ALFA/LABDA VALUE

1. **Split and Standardize the data** (only model inputs and not the output)
2. **Decide which regression technique** Ridge, Lasso, or Elastic Net you wish to perform (or test all of them)
3. **Use GridSearchCV** to optimize the hyper-parameter alpha, and lambda ratioS in elastic net
4. **Build your model** with your now optimized alpha and make predictions!

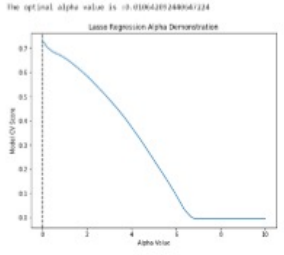


Fig from sklearn,

FEATURE SELECTION WITH REGULARIZATION

- **LassoCV** is most often preferable for high-dimensional datasets with many collinear features
- **LassoLarsCV** allows exploring more relevant values of alpha parameter then Lasso method, and, if the n<<p, it is often faster than LassoCV.

COMMENT: should You even use Lasso regression?

Comments that I found on stack overflow: Even in the case when you have a strong reason to use L1, given the number of features, I would recommend going for Elastic Nets instead. Granted this will only be a practical option if you are doing linear/logistic regression. But, in that case, Elastic Nets have proved to be (in theory and in practice) better than L1/Lasso. Elastic Nets combine L1 and L2 regularization at the "only" cost of introducing another hyperparameter to tune (see Hastie's paper for more details Page on stanford.edu). Even in a situation where you might benefit from L1's sparsity in order to do feature selection, using L2 on the remaining variables is likely to give better results than L1 by itself.
I do not agree, there are specific applications where L1 regr., is preferred, such as for feature selection eg: in genetics, gene expression studies, signal processing

What else beside regularization can be used to improve your model?

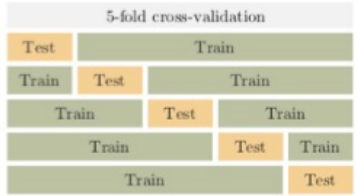
APPLY PROPER DASTA TRANSFORMATIONS

- Scaling/normalization/
- Binarization/Kbins selection
- Non-linear tranformations
- Add polynomial features,

MODIFY SAMPLING STRATEGY

- Data Augmentation
- Synthetic Data
- K-fold Cross-Validation (CV)

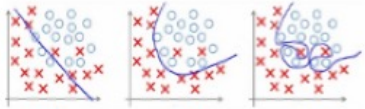
Divide the data into k groups. Train on (k-1) groups and test on 1 group. Try all k possible combinations.



USE MODEL PARAMETERS TO REDUCE. VARIANCE

E1. k-nearest neighbours (k-nn)

Generally k-nn alg has low bias & high variance, but the trade-off can be changed **by increasing the k value** which increases the number of neighbours that contribute to the prediction and in turn increases the bias of the model.



E 2. The support vector machine (SVM)

has low bias and high variance, but the trade-off can be changed by **increasing the C parameter** that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

MODIFY TRAINING APPROACH

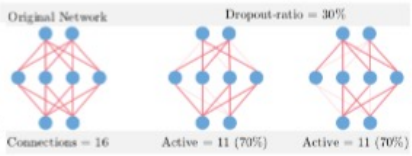
Noise Injection

- Add random noise to the weights when they are being learned.
- It pushes the model to be relatively insensitive to small variations in the weights, hence regularization

Dropout

- Generally used for neural networks.
- Connections between consecutive layers are randomly dropped based on a dropout-ratio and the remaining network is trained in the current iteration. In the next iteration, another set of random connections are dropped.

Pruning

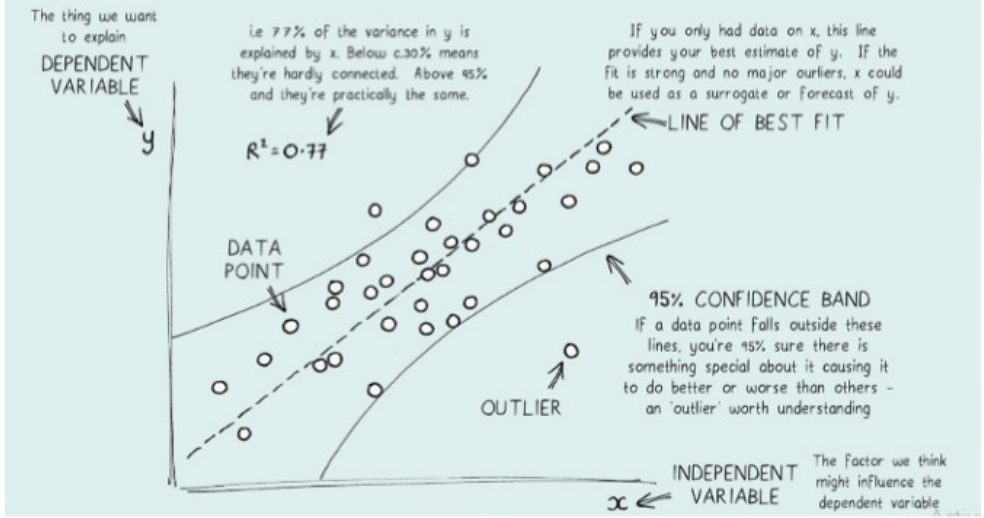


by **Pawel Rosikiewicz**

Linear vs Non-Linear Regression Models

www.SimpleAI.ch

LINEAR REGRESSION



NON-LINEAR REGRESSION

Simply - If a regression equation doesn't follow the rules for a linear model (all elements are either constant or parameter multiplied by the term, and then addended to each other), then it is a nonlinear model.

PROS

Non-linear regression model can fit an enormous variety of curves. - However, because there are so many candidates, you may need to conduct some research to determine which functional form provides the best fit for your data.

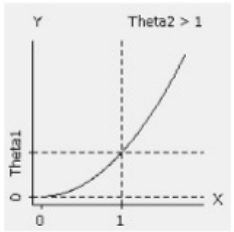
CONS

- Can not use r'2 scores
- Can not calculate p-values for the terms

EXAMPLES OF NON-LINEAR REGR. MODELS

IMPORTANT: each function can fit a variety of shapes, and there are many nonlinear functions. Also, notice how nonlinear regression equations are not comprised of only addition and multiplication!

** thetas are the parameters, and Xs are the independent variables.*

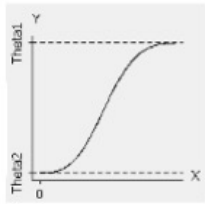


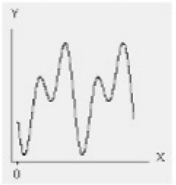
Power:

$$\theta_1 * X^{\theta}2$$

Weibull growth:

$$\theta_1 + (\theta_2 - \theta_1) * exp(-\theta_3 * X^{\theta}4)$$





Fourier:

$$\theta_1 * cos(X + \theta_4) + \theta_2 * cos(2 * X + \theta_4) + \theta_3$$

See also
Gradient Boosting Regression Trees

Which one to choose?

First use linear regression, and determine whether it can fit the particular type of curve in your data. If you can't obtain an adequate fit using linear regression, that's when you might need to choose nonlinear regression.

- **Pros:** Linear regression is easier to use, simpler to interpret, and you obtain more statistics that help you assess the model.
- **Cons:** While linear regression can model curves, it is relatively restricted in the shapes of the curves that it can fit.

Then, Use Nonlinear regression that can fit many more types of curves, but it can require more effort both to find the best fit and to interpret the role of the independent variables. Additionally, R-squared is not valid for non-linear regr., and it is impossible to calculate p-values for the parameter estimates.

ATTRIBUTES OF LINEAR REGR.

We assume linear relationship between input features and response variables, eg: if we increase value a by certain about, there will be also a, increase or decrease of the response variable,

- A linear regression model follows a very particular form.
  $y = w0 + w_1x_1 + w_2x_2 + ... + w_nx_n$
- A regression model is linear when all terms in the model are one of the following:
  - The constant, denoted as "c", or, "w_0"
  - A parameter multiplied by an independent variable
    eg w_1, w_2, w_3
  - Then you need to simply add all the terms to each other.
    $y = w0 + w_1x_1 + w_2x_2 + ... + w_nx$
- **this type of regression equation is linear in the parameters.**

FITTING CURVED LINES WITH LINEAR REGR.

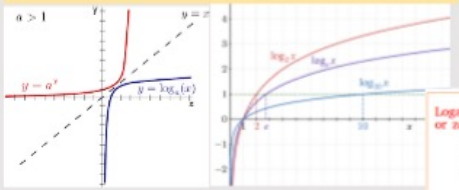
polynomial regr.

- The function must be linear in the parameters (see above), you can raise an independent variable by an exponent to fit a curve. For example, if you square an independent variable, the model can follow a U-shaped curve.
  $Y = w0 + w_1x_1 + w_1x_1^{\wedge}2$
- While the independent variable is squared, the model is still linear in the parameters.

Other term modifications, such as **log terms and inverse terms** can also be used to follow different kinds of curves and yet continue to be linear in the parameters.

More info in Sklearn
https://inria.github.io/scikit-learn-mooc/python_scripts/linear_regression_non_linear_link.html

Exponential f. vs log function



Logarithm of a negative number or zero

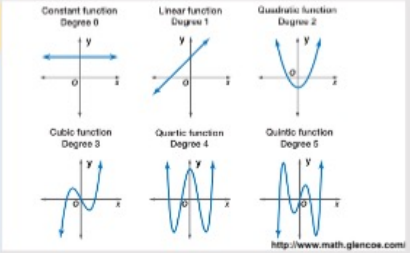
$\log_{10}$ -20 = undefined
$\log_{10}$ -6 = undefined
$\log_{10}$ 0 = undefined

Exponential f. vs power function

Exponential f. $f_{(x)} = 3^x$
Power f. $f_{(x)} = x^{\wedge}3$

Power functions vs Polynomial f.

Polynomial function is the sum of terms, each of which consists of a **transformed power function** with non-negative integer powers. The degree of a polynomial function is the highest power of the variable that occurs in a polynomial



SELECTED SOURCES
some materials taken rom:
https://statisticsbyjim.com/regression/difference-between-linear-nonlinear-regression-models/
https://bookdown.org/tpinto_home/Beyond-Linearity/polynomial-regression.html

REGRESSION - intro

Highly interpretable method, for modelling past relationships (x->y), or to help predict output variables (y) based on the new input var's (x).

- Understand product-sales drivers eg: competition prices, distribution, advertisement,
- Optimize price points and estimate product-price elasticities

Step 1: Importing the required Libraries

These Two are essential libraries which we will import every time.
NumPy is a Library which contains Mathematical functions.
Pandas is the library used to import and manage the data sets.

Step 2: Importing the Data Set

Data sets are generally available in .csv format. A CSV file stores tabular data in plain text. Each line of the file is a data record. We use the read_csv method of the pandas library to read a local CSV file as a dataframe. Then we make separate Matrix and Vector of independent and dependent variables from the dataframe.

Step 3: Handling the Missing Data

The data we get is rarely homogeneous. Data can be missing due to various reasons and needs to be handled so that it does not reduce the performance of our machine learning model. We can replace the missing data by the Mean or Median of the entire column. We use Imputer class of sklearn.preprocessing for this task.

Step 4: Encoding Categorical Data

Categorical data are variables that contain label values rather than numeric values. The number of possible values is often limited to a fixed set. Example values such as "Yes" and "No" cannot be used in mathematical equations of the model so we need to encode these variables into numbers. To achieve this we import LabelEncoder class from sklearn.preprocessing library.

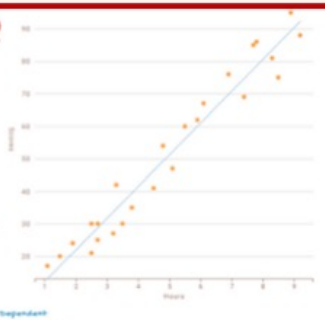
Step 5: Splitting the dataset into test set and training set

We make two partitions of dataset one for training the model called training set and other for testing the performance of the trained model called test set. The split is generally 80/20. We import train_test_split() method of sklearn.crossvalidation library.

Step 6: Feature Scaling

Most of the machine learning algorithms use the Euclidean distance between two data points in their computations. features highly varying in magnitudes, units and range pose problems. High magnitudes features will weigh more in the distance calculations than features with low magnitudes. Done by Feature standardization or Z-score normalization. StandardScalar of sklearn.preprocessing is imported.

SIMPLE LINEAR REGRESSION



Predicting a response using a single feature.

It is a method to predict dependent variable (Y) based on values of independent variables (X). It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

$$y = b_0 + b_1 x_1$$

(dependent variable; independent variable)

How to find the best fit line?

In this regression model, we are trying to minimize the errors in prediction by finding the "line of best fit" – the regression line from the errors would be minimal. We are trying to minimize the length between the observed value (Yi) and the predicted value from our model (Yp).

(observed value; predicted value)

$$y_i \quad y_p$$

$$\min \{SUM(y_i - y_p)^2\}$$

In this regression task, we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied.

(slope; y - intercept)

$$Score = b_0 + b_1 * hours$$

STEP 1: PREPROCESS THE DATA

We will follow the same steps as in my previous infographic of Data Preprocessing.

- Import the Libraries.
- Import the DataSet.
- Check for Missing Data.
- Split the DataSet.
- Feature Scaling will be taken care by the Library we will use for Simple Linear Regression Model.

STEP 2: FITTING SIMPLE LINEAR REGRESSION MODEL TO THE TRAINING SET

To fit the dataset into the model we will use LinearRegression class from sklearn.linear_model library. Then we make an object regressor of LinearRegression Class. Now we will fit the regressor object into our dataset using fit() method of LinearRegression Class.

STEP 3: PREDICTING THE RESULT

Now we will predict the observations from our test set. We will save the output in a vector Y_pred. To predict the result we use predict method of LinearRegression Class on the regressor we trained in the previous step.

STEP 4: VISUALIZATION

The final step is to visualize our results. We will use matplotlib.pyplot library to make Scatter Plots of our Training set results and Test set results to see how close our model predicted the Values

MULTIPLE LINEAR REGRESSION

Multiple linear regression attempts to model the relationship between two or more features and a response by fitting a linear equation to observed data. The steps to perform multiple linear regression are almost similar to that of simple linear regression. The difference lies in the evaluation. You can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other.

(dependent variable; multiple independent variables)

$$y = b_0 + b_1 x_1 + b_2 x_2 \dots \dots b_n x_n$$

ASSUMPTIONS

FOR A SUCCESSFUL REGRESSION ANALYSIS, IT'S ESSENTIAL TO VALIDATE THESE ASSUMPTIONS.

1. Linearity: The relationship between dependent and independent variables should be Linear.
2. Homoscedasticity (constant variance) of the errors should be maintained.
3. Multivariate Normality: Multiple regression assumes that the residuals are normally distributed.
4. Lack of Multicollinearity: It is assumed that there is little or no multicollinearity in the data. Multicollinearity occurs when the features (or independent variables) are not independent of each other.

NOTE

Having too many variables could potentially cause our model to become less accurate. especially if certain variables have no effect on the outcome or have a significant effect on other variables. There are various methods to select the appropriate variable like -

1. Forward Selection
2. Backward Elimination
3. Bi-directional Comparision

DUMMY VARIABLES

Using categorical data in Multiple Regression Models is a powerful method to include non-numeric data types into a regression model.
Categorical data refers to data values which represent categories - data values with a fixed and unordered number of values, for instance, gender (male/female). In a regression model, these values can be represented by dummy variables - variables containing values such as 1 or 0 representing the presence or absence of the categorical value.

DUMMY VARIABLE TRAP

The Dummy Variable trap is a scenario in which two or more variables are highly correlated, in simple terms, one variable can be predicted from the others. Intuitively, there is a duplicate category: if we dropped the male category it is inherently defined in the female category (zero female value indicate male, and vice-versa)
The solution to the dummy variable trap is to drop one of the categorical variables - if there are m number of categories, use m-1 in the model, the value left out can be thought of as the reference value.

(Dummy variable; Dummy variable)

$$D_2 = 1 - D_1$$

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_3 D_1$$

1 PREPROCESS THE DATA

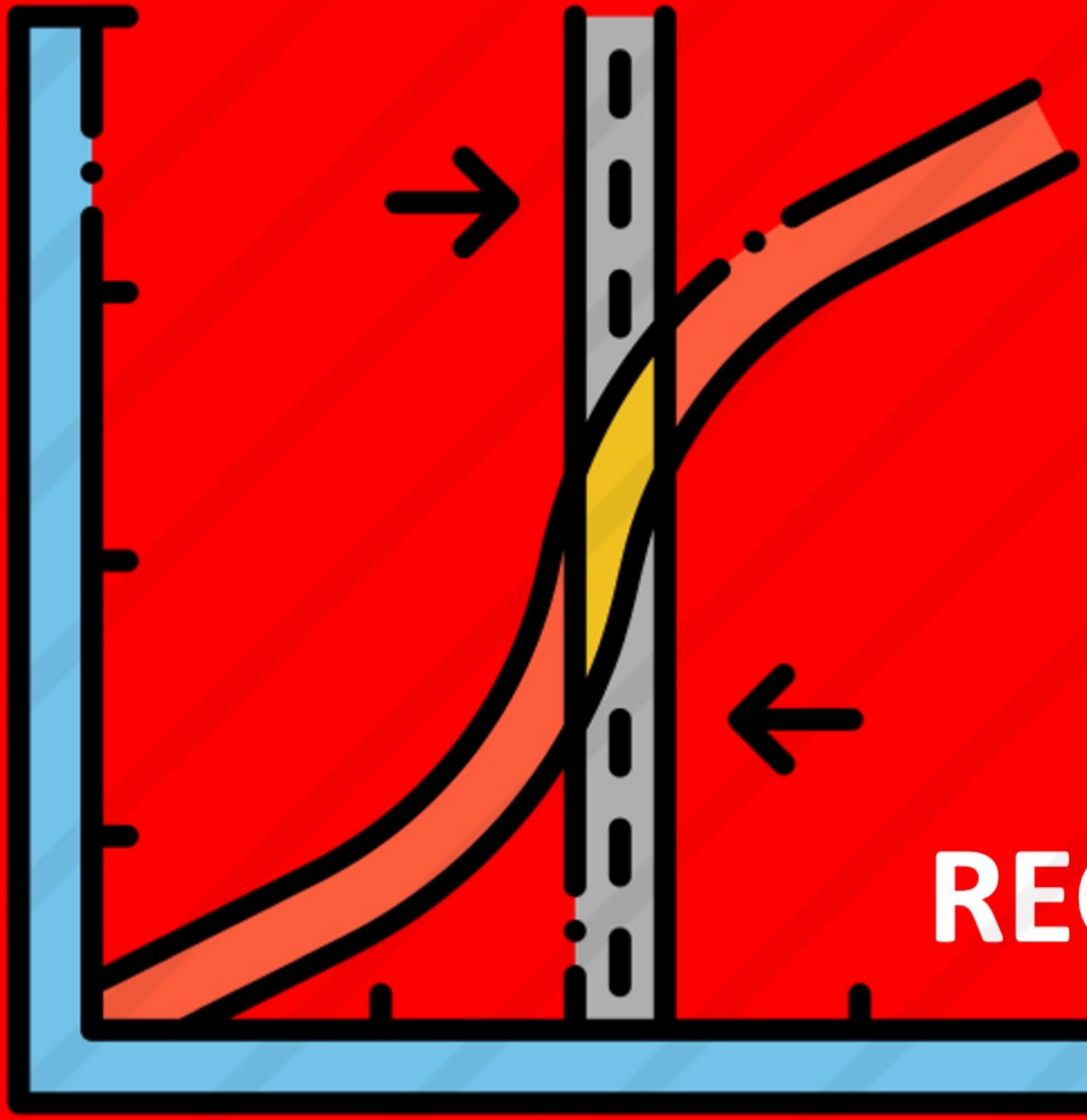
- Import the Libraries.
- Import the DataSet.
- Check for Missing Data.
- Encode Categorical Data
- Make Dummy Variables if necessary and avoid dummy variable trap.
- Feature Scaling will be taken care by the Library we will use for Simple Linear Regression Model.

2 FITTING OUR MODEL TO THE TRAINING SET

This step is exactly the same as for simple linear regression. To fit the dataset into the model we will use LinearRegression class from sklearn.linear_model library. Then we make an object regressor of LinearRegression Class. Now we will fit the regressor object into our dataset using fit() method of LinearRegression Class.

3 PREDICTING THE TEST RESULTS

Now we will predict the observations from our test set. We will save the output in Y_pred. To predict the result we use predict() method of LinearRegression Class on the regressor we trained in the previous step.



LOGISTIC REGRESSION

'image: Flaticon.com'.

by **Pawel Rosikiewicz**

Logistic Regression vs Linear Regression

SimpleAI.ch

Linear regression
(for comparison)
predicts continuous numerical output variable

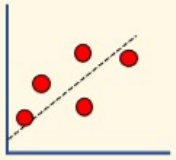
ASSUMPTIONS

- ***Validity of Linear Models*** *(linear relationship between X and y)*
- ***No multicollinearity in the data***
- ***Normally distributed residuals***
- ***Homoscedastic residuals*** */ constant variance*
- ***No auto-correlation*** */ independence of errors*

MODEL

INPUT DATA
- *Continuous, Discrete,*
- *Categorical, Ratio, Binary*

OUTPUT DATA
- *Continuous numeric variable, from –inf to +inf*

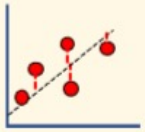


$$y = w0 + w_1x_1 + w_2x_2 + ... + w_nx_n$$

- **Linear regression;** fits the line/hyperplane by minimizing RSS, MSE; or MAE error functions
- **Multiple linear regression**

MODEL FITTING

The model is fitted with different methods eg OLS, or SGD, by minimizing ***the residuals,***



MODEL EVALUATION & SELECTION
- Trian/test errors,
- R^2 (problems with R^2 expansion)
- p values
- Estimate Model complexity (eg. weight values, smaller are better)
- Test model assumptions

OPTYMIZATION

SOLVERS
- **OLS** (closed form)
- **GD, SGD, or minibatch**
- **SubGradient Descent** (non-continuous cost functions, like MAE)
- **MLE** – max likelihood Estimator, only for MSE

OVERFITTING
- Apply Regularization,
- get more data,
- Simplify the model

UNDER FITTING
- Polynomial expansion,
- Feature engineering
- F. selection

Logistic regression
Predicts P(y=1|X)

icons: Flaticon.com

ASSUMPTIONS

- ***Binary target variable*** *(but not always)*
- ***Observations must be independent from each other*** *(no autocorrelation)*
- ***Linear relationship between independent variables and target log odds***
- ***No multicollinearity in the data***

IT DOES NOT NEED
- ***Dependent and independent variables do not have to have linear relationship***
- ***THERE ARE NO RESIDUALS !*** *Thus, no assumptions on normally distributed, homoscedastic, and independent residuals may be used*

BEST IF
- ***Best if Input data have multivariate normal distribution***
- ***No strongly influential outliers*** *(eg test with cook's distance)*
- ***Typically, it requires a large sample size:*** *Rule of Thumb: have min. 10 cases with the least frequent outcome for each independent variable eg: if you have 5 independent var's and the expected P of your least frequent outcome is .10, then you would need a minimum sample size of 500 (10*5 / .10).*

Logistic regression in Python (imbalanced classes) https://machinelearningmastery.com/multi-class-imbalanced-classification/

Logistic regression in R (testing assumptions) http://www.sthda.com/english/articles/36-classification-methods-essentials/148-logistic-regression-assumptions-and-diagnostics-in-r/

MODEL

LOGREG MODEL HAS THREE MAIN COMPONENTS

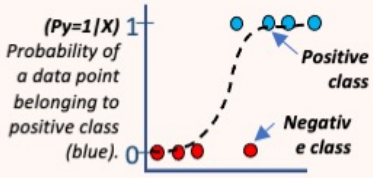
(i) **Linear Predictor;** logreg can fit either a line, or polynomial with sigmoid activation f.

(ii) **Link function;** "g", mapping the values returned by linear predictor onto the distribution. logreg, uses specific type of a sigmoid function

(iii) **Probability distribution from an exponential family,** it used to make prediction on error distribution and to find optimal fitted line, with MLE. Logreg uses logistic function that have a higher kurtosis then normal distrib. (wider tails)

INPUT DATA
- *Continuous, Discrete,*
- *Categorical, Ratio, Binary*

OUTPUT DATA
- *the probability of a data point being in the positive class: p(y=1|x)=sigma(fx), where f(x) is a linear regr. fn on input data and logit values, sigma is a binary target variable ∈{0,1}.*



SIMILAR MODELS

Ie. GLM models that use different link function, and distribution for MLE
- **POISSON R;** for count data.
- **LOGISTIC & PROBIT R;** for binary data
- **MULTINOMIAL LOGISTIC & PROBIT R;** for cat. data.
- **ORDERED LOGISTIC & PROBIT R.;** for ordinal data

MODEL FITTING

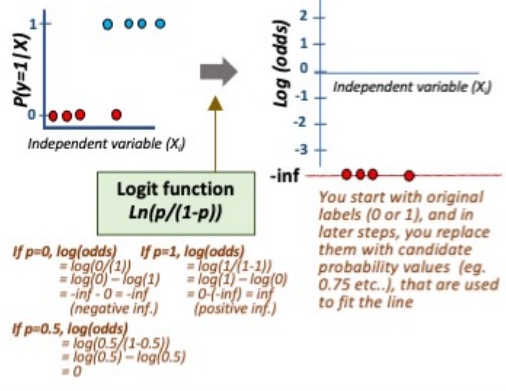
These are the 4 idealistic steps used to fit logreg model to the data, and they approximate what happens in the 1st iteration step

Step 1. Transform P(y=1|X) into log(odds) with logit function.

Step 2. Draw the candidate line, & project candidate log(odds) *values on that line*

Step 3. Transform candidate log(odds) into candidate prob. with sigmoid function

Step 4. Calculate likelihood for all the data points



If p=0, log(odds) = log(0/(1)) = log(0) – log(1) = -inf - 0 = -inf (negative inf.)

If p=1, log(odds) = log(1/(1-1)) = log(1) – log(0) = 0-(-inf) = inf (positive inf.)

If p=0.5, log(odds) = log(0.5/(1-0.5)) = log(0.5) – log(0.5) = 0

You start with original labels (0 or 1), and in later steps, you replace them with candidate probability values (eg. 0.75 etc..), that are used to fit the line

- Linear predictor -
This is the line/hyperplane used to predict new y values
$Log(odds) = w0 + w_1x_1 + ... + w_nx_n$

sigmoid function
$$p = \frac{e^{log(odds.)}}{(1 + e^{log(odds)})}$$

Distance between inf, and its projection on the line, is also inf, thus they can't be used as residuals == NO RSS/OLS methods for optimization

This, S–shape, dashed line comes from log(odds), not from values on x-axis. There is a connection between them in step 2, but not in step 3, except for mapping specific p to specific X values

Likelihood of y=1|X =
0.25 *0.81 *0.94 *0.97

Likelihood of y=0|X =
(1-0.02) *(1-0.1) *(1-0.32) *(1- 0.67)

Total likelihood =
*Likelihood y=1|X * Likelihood y=0|X*

Applying log, we can sum them up, and use to find best fitting line !

MODEL EVALUATION & SELECTION
- Trian/test Acc, Recall, Sensitivity,
- ROC & PR curves, AUC
- R^2 (calculated differently)
- p values
- Log-loss

LOGISTIG REGRESSION: OPTYMIZATION

LOSS FUNCTION
- Defined with binomial prop. for each point
- Using probabilities, returned by sigmoid fn.
- After defining loss function, you must take its partial derivatives, for each parameter (weight) and update them, with step size (go up, not down!)

$$\text{Log Loss} = \sum_{(x,y)\in D} -y\log(y') - (1-y)\log(1-y')$$

REGULARIZATION

Its very important in logreg, especially in high-dimensional data, where rare combinations of features, would drive model to overfitting, and increase coeff. toward inf. REMEMBER TO SCALE THE DATA
- **Early stopping**
- **L2 regularization**
- **ElasticNet**
- **L1, can also be used with sparse data**

SOLVERS / SKLEARN

Remember: *each solver works only with a subset of penalty terms, and some, like liblinear can only use OvR strategy for multiclass classif.*
- ***Lbfgs –*** *efficient with small number of samples, fast convergence eg maxiter =100*
- ***SAG & SAGA for*** *large datasets, Stochastic average gradient, and its modiff, SAGA*
- **newton-cg,** ; for large, sparse data

MULTILABEL CLASSIF.
- Softmax Regr;
- OvO; one vs One,
- OvR; one vs rest

in multiclass classf, skleanr uses Cross-entropy cost function and cdf of multinomial cdf

KEY TAKEAWAYS
- *We predict probability of one class, given the data X P(y=1|X)*
- *You only transform binary target variable, y to log(odds), not the X*
- *Log(odds) are used instead of probability to allow applying linear predictor function, and fits its line to the data*
- *When predicting new values, logreg, returns Log(odds) which are, subsequently transformed into probabilities with sigmoid function, to provide us a meaningful results. Thus, the probabilities are not coming directly from X, but are mapped onto them*

by **Pawel Rosikiewicz**

Logistic R. vs Linear R. - cost function & optimization with MLE

SimpleAI.ch

MLE: maximum Likelihood for Linear Regression (recap) Estimator

Here; I followed lectures of Martin Jaggi @EPFL:
https://youtu.be/fZELYM4ePDM

Step 1. Define the assumptions
The most important is to decide what is the distribution of errors. That equation will be used to calculate likelihood of the observed data, given the inputs, and the model with its parameters

MLE assumption for Linear regression:

(i) The data were generated with some true/unknown underlying model $y_n = x_n^T w + e_n$

(ii) e, error – it is a zero-mean gausian random variable added independently to each sample,

(iii) The samples itself are i.d.d (independent rand. Variables). Therefore, the total likelihood of the results observed, guven the data, and random error e, can be calculated as muliplication

$p(y_n|x_n, w) \sim N(y_n | x_n^T w, sigma^2)$
$P(e_n) = N(e_n | 0, sigma^2)$

.... and our goal is to **maximize that likelihood**

Therefore, given N samples, the likelihood of a data vector $y = (y1, ...yn)$, given the input X, and the model w, is:

model $\mathbf{w}$ is equal to

likelihood — independence — assumption on E_n

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(y_n \mid \mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}(y_n \mid \mathbf{x}_n^\top \mathbf{w}, \sigma^2).$$

Step 2. Define cost with Log-Likelihood (LL cost f.)

- *Essentially, you replace the values in equation of a normal distribution with residuals, (true_y - predicted_y)*
- *Next, you replace y_predicted with the model, and input data.*
- *calculate likelihood for each point, and multiply them – it can only be done if the samples are independent from each other.*

Use Log Likelihood to avoid multyplying small numbers and keep precission (in norm. distrib it will remove exp. function)

$\log p(y|x, w)$
$= \log {}_{n=1}\Pi^N \, p(y_n|x_n, w)$
$= \log {}_{n=1}\Pi^N \, N(y_n | x_n^T w, sigma^2)$

This becomes constant, and can later be ignored, because it is independent from the weights.

replace values in Gaussaian distribusion with residual, that we assumed have normal distribution in step 1.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$= \log {}_{n=1}\Pi^N \; const \cdot exp\,(- (y_n - x_n^T w)^2 / 2\, sigma^2)$

Finally, we are only interested to find the weights that maximize the likelihood, thus we can replace cost, exp, and sigme^2 with new const' that is often ignored, as independent from weights in the model

$= -{}_{n=1}\Sigma^N (y_n - x_n^T w)^2 + const'$

Therefore, in short

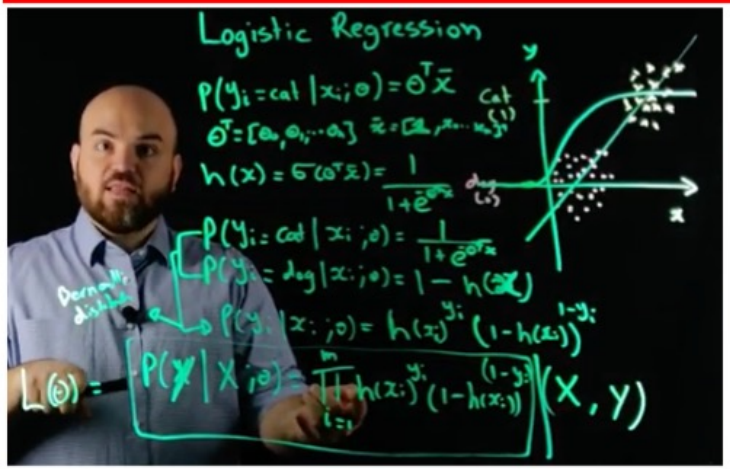
$L_{LL}(w) := \log p(y|x, w) = -(2sigma^2)^{-1} \, {}_{n=1}\Sigma^N (y_n - x_n^T w)^2 + const'$
|---------- | likelihood

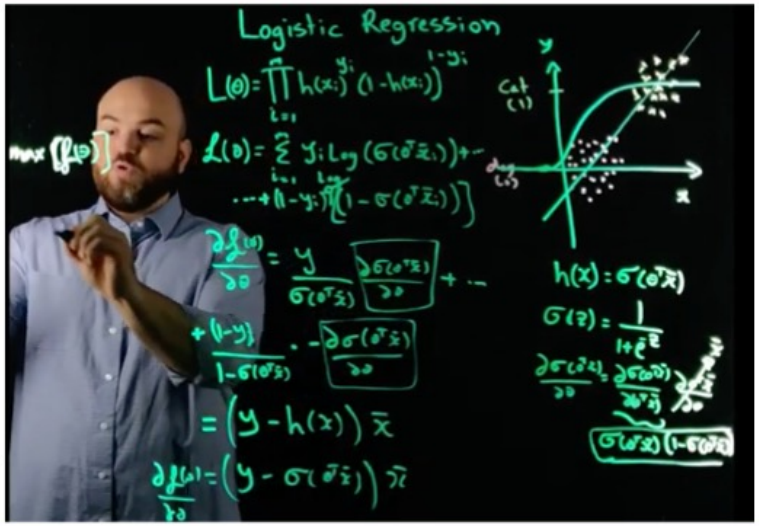
Constant in the equation (const) – is it a problem?
No, It is a value independent of w, thus it does not matter in practice (all compared likelihood values will have the same const added)

How do you know the variance in the model?
You may use variance from empirical data. However, it is only used as scaling factor, so as long as it is constant, it will not affect the results

exp(x) = e^x
eg: exp(2) = 7.38..., exp(0.5) = 1.64..., exp(-2)=0.13...,

MLE: maximum Likelihood Estimator for Logistic regression





Endless Engineering — Slides from youtube channel: Endless Engineering

Step 3 Optimization
We need to get derivatives, for each weight (w), and then we can use GD or SGD to optimize them with step size, Cation! you ascent with the gradient

Challenges

- ***We do not know whether each L_{LL} is differentiable and convex.***
- ***The above solution is very specific.*** *If you change assumptions, eg: on the error distribution, or independence you would have to use different equation, or more complicated functions.*
- ***Strong assumptions*** *– in reality, we shouldn't, or can not assume that all errors comes form uniform and homoscedastic normal distribution. Yet, it is a good approximation*

L_{LL} vs MSE

log likelihood — indep. of w

$$\mathcal{L}_{\mathrm{LL}}(\mathbf{w}) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \mathbf{x}_n^\top \mathbf{w})^2 + \mathrm{const}$$

→ max

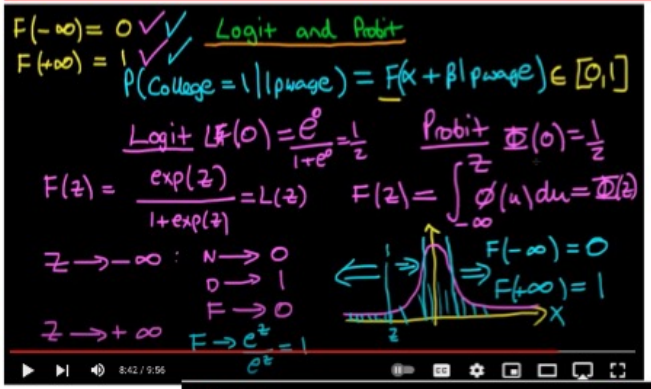
$$\mathcal{L}_{\mathrm{MSE}}(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - \mathbf{x}_n^\top \mathbf{w})^2$$

cost (mse) → min

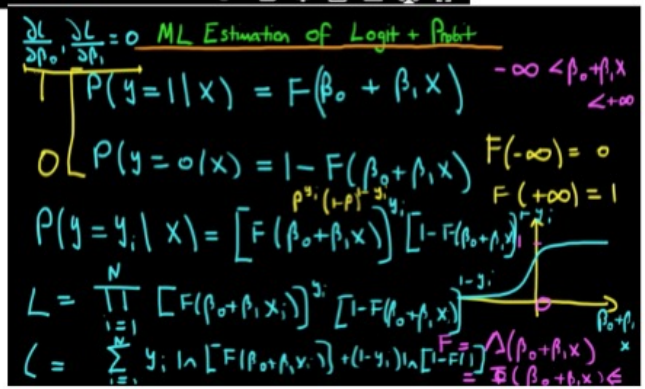
Pros & cons of MLE

- *allows to make the connection between real underlying model, the data, and the estimations made with evaluated model.*
- *It can work with NOT NORMALLY DISTRIBUTED error or data, ie. you may use different distrib's to build max. likelihood function.*
- *requires to make several assumptions, eg: data generated in rand. process, i.d.d, noise has norm. distr. with mean=0.*
- *The solution is more complicated then by minimalizing Loss function and provides similar weights.*

MLE for Probit and Logreg



Slides from youtube channel: Khans academy



by **Pawel Rosikiewicz**

EDA & DATA PREPARATION FOR LOGISTIC REGRESSION

www.SimpleAI.ch

MODEL ASSUMPTIONS

- ***Binary target variable** (but not always)*
- ***Observations must be independent from each other** (no autocorrelation)*
- ***Linear relationship between independent variables and target log odds***
- ***No multicollinearity in the data***

IT DOES NOT NEED

- ***Dependent and independent variables do not have to have linear relationship***
- ***THERE ARE NO RESIDUALS !** Thus, no assumptions on normally distributed, homoscedastic, and independent*

BEST IF

- ***Best if Input data have multivariate normal distribution***
- ***No strongly influential outliers** (eg test with cook's distance)*
- ***Typically, it requires a large sample size:** Rule of Thumb: have min. 10 cases with the least frequent outcome for each independent variable eg: if you have 5 independent var's and the expected P of your least frequent outcome is .10, then you would need a minimum sample size of 500 (10*5 / .10).*

MODEL EVALUATION & SELECTION

- Trian/test Acc, Recall, Sensitivity,
- ROC & PR curves, AUC
- R^2 (calculated differently)
- p values
- -Log-loss (ie. mean -log loss of corrected, predicted probabilities,

PREPARE THE DATA

FORMAT & CLEAN THE OUTPUT VARIABLE

CREATE BINARY TARGET VARIABLE

you may use multiclass classification too with skleatn, with 2 different strategies. Remember, that OvR is faster then OvO approach,

REMOVE NOISE IN THE TARGET V.:

Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.

SCALING & CLEANING IMPUT DATA

STANDARIZE/SCALE INPUT V.

Standardizing, or scaling, is especially important for SAG and SAGA algorithms, but it may improve model accuracy, optimized with the other methods.

DETECT OUTLIERS & REMOVE THEM

- See my slides on outlier detection

ENSURE INPUT DATA HAVE MULTIVARIATE NORMAL D.

-- TESTING --

VISUAL INSPECTION

- ***Histogram***
- ***QQ-Plots** (Quantilie-Quantile Plots), or **PP-Plots** (Probability Plots)*
- ***Boxplot***

DESCRITIVE STATISTICS

- ***Mean;** should be 0,*
- ***Skewness;** should be 0*
- ***Kurtosis;** ideally ~2.2*

STATISTICAL TEST FOR NORMALITY

- **Shapiro–Wilk test** (small and medium size datasets)
- **Scipy normaltest** (up to~50 samples);
- **Lilliefors-test** (50-300 samples),
- **Kolmogorov-Smirnov** (>300 samples)

-- HANDLING --

TRANSFOMR INPUT VARIABLES

- ***log, root, reprpicotal***
- ***Quantile transformer**, a non-parametric approach*
- ***Power transformer***
  - ***'box-cox'** – positive data only*
  - ***'Yeo-Johnson'** – people and negative values dat.*

TEST ASSUMPTIONS

LINEARITY ASSUMPTION X~LOG(ODDS(Y))

There is a linear relationship between the logit of the outcome and each predictor variables, where logit(p) = log(p/(1-p))

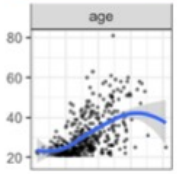
-- TESTING --

VISUAL INSPECTION & CORRELATION ANALYSIS

In general, you must first fit your model to traning, data, and then, plot predicted log(odds) values for train, or test data against, each feature, with scatterplot, and trendline. Personally, I rarely use that approach, but it may be important, if you use feature selection, and you wish to examine quantitative features.

>>> sns.regplot(); sns.lmplot

https://seaborn.pydata.org/tutorial/regression.html#regression-tutorial



-- HANDLING --

TRANSFOMR INPOUT VARIABLES

- *Polynomial expansion,*

ENSURE NORMAL DISTRIB.

NO MULTICOLINEARITY

BEFORE YOU START

Remove one column in each one-hot-encoded feature

-- TESTING --

Pair-wise comparisons (X);

find which pairs of predictor variables are correlated with each other.**Use :**

- ***Pearson corr.** (linear),*
- ***Spearmen corr.** (non-linear).*
- ***Heatmap**, or table with top results (use absolute values, or head/tail)*

One vs Rest comparisons (X);

Used to find which predictor variable is correlated with many other predictors in the input dataset.

- ***Variance Inflation Factor (VIF)** one feature is used as dependent var. and all other features as its predictors*

-- HANDLING --

Manually Remove highly corr. features

- *Corrs ~-1 or 1*
- *VIF > 5*

Use regularization to remove corr features

- *LASSO R.*
- *Spearmen corr. (non-linear).*

Use hierarchical clustering to remove correlated features,

- *eg: Compute spearman rank order coeff. and pick a single feature from each cluster of features correlated with each other*

Dimensionality reduction

- *take the top eigenvectors that preserve the max. variance*

NO AUTOCORRELATION

BEFORE YOU START

Reverse probability values for one class (1-P)

Or use only one, class at the time.

-- TESTING --

RUN CHART:

Plot probability results vs time/spatial variable, typically, line plot to visualize the pattern.

- ***Sample order – first suspect,***
- *Time variables*
- *Spatial variables,*

LAG PLOT/CORRELOGRAM;

scatter plot with the two variables (X, Xlag). X"lag" is a the same variable shifted by the fixed amount of passing time/order; you may use ... see list in the above

DURBIN-WHATSON TEST

measures the amount of autocorrelation in residuals from the regression analysis, can be adapted to checks ONLY for the first-order autocorrelation, ie. lag =1

-- HANDLING --

Improve the model

- *Add new spation/timedependent variable*
- *Play with regularization strength – caution sklearn uses C, that is inverted alpha value, ie. the higher C, then lower the strength of the regularization, typically we use C from0.01 to 100*
- *VIF > 5*

Feature transformation

Transforming var's & test if the autocorrelations were reduced.

- deviation from the average values; eg: weather data.
- Log or exponential - may or may not make improvements.
- Annualising - to remove seasonal eff

Use hierarchical clustering on the time invariant factors

- ***Ie, cluster the data based on features, other then, the features causing autocorrelation,***

Logistic vs Probit regression

- *Generally, both models yield similar results, with some exceptions,*
  - *If a multiclass classification, have IIA assumption, that is violated, the probit model deals with it better.*
  - *Coeeficients probit regr. are rescaled of factor of ~1.6, in comparison to logistic regression, but similar among model weights*
- Logit model is much easier to deal with analytically, ie there is no integrals – probit use them, because it is based on normal distribution and its cdf is defined with integrals.
- log(odds) are easier to interpret,
- *probits are always between 0 and 1, and the probate incorporates non-linear effects of X as well.*

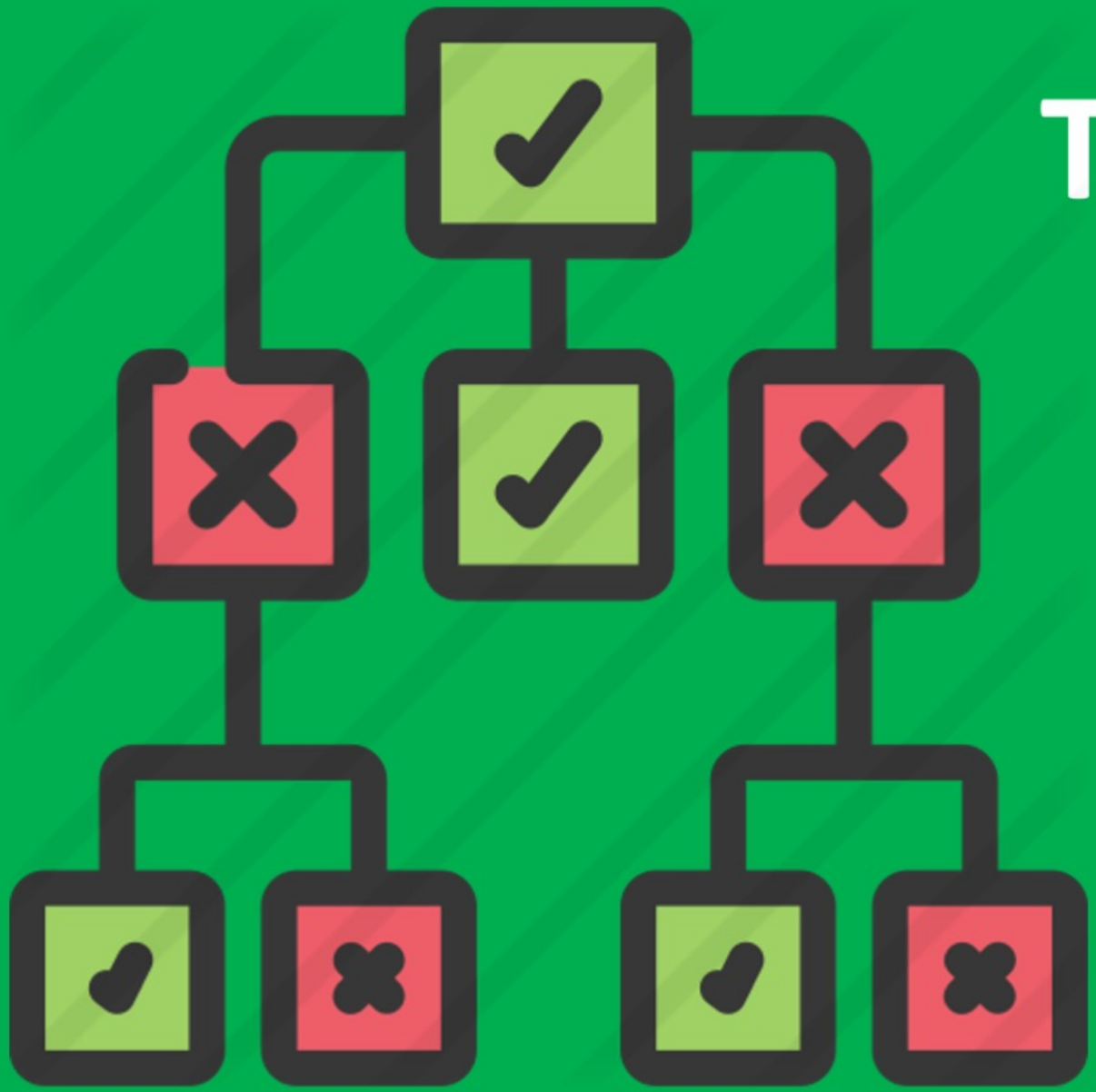
Handling Problems

CLASS IMBALANCE

- Target variable transformation: log, sqrt or box-cox
- WLS, or other type of weights

MULTICLASS CLASSYFICATION

- Target variable



TREE BASED METHODS

'image: Flaticon.com'.

by Pawel Rosikiewicz

CLASSYFICATION TREES

www.SimpleAI.ch

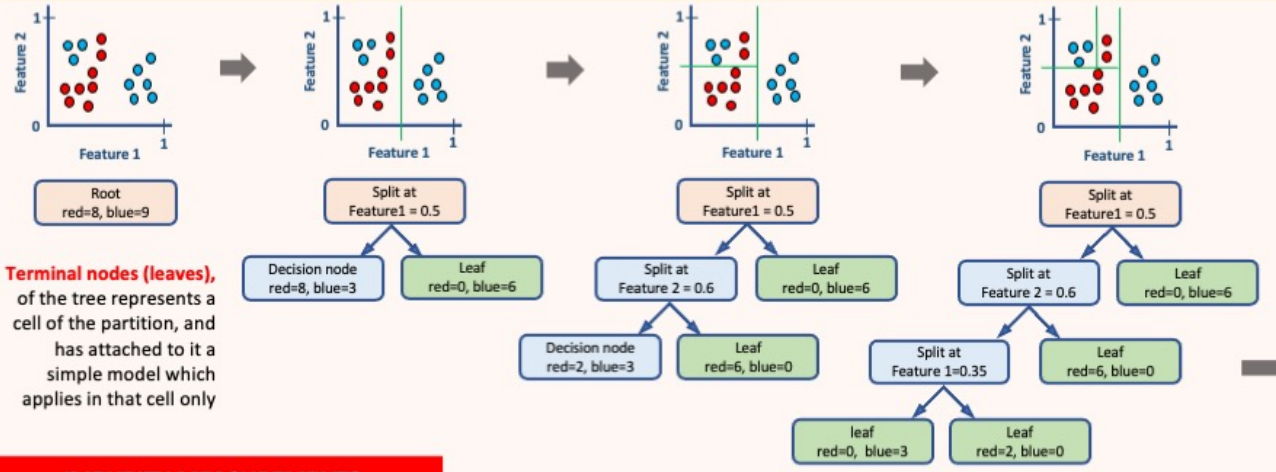
MODEL ASSUMPTIONS

- ***all data points have common root,*** *and can be divided into smaller subsets using if/else rules, like >0 is positive, <0 is negative*
- ***decision tree model is hierarchical,*** *ie the order to decisions and features have meaning,*
- ***DT follow Sum of Product (SOP) representation (Disjunctive Normal Form).*** *- For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum).*
- **DT do not make assumptions on**
  - Response variable distribution
  - Error distribution
  - Can work on discontinuous and not smooth feature space
  - do not assume a particular form of relationship between the independent and dependent variables

HOW DECISION TREE ALG's WORKS

The recursive partitioning algorithms

- Same as in hierarchical clustering, but with labelled data that can be used to select the best criteria of each split - impurity measurement
- Trees are local models, they are recursively partitioning the space, forgetting about the previous decisions.
- In a given branch, a new split will be made based on what partitions the data best in the corresponding part of the space, regardless of the criteria that were used to make the previous splits

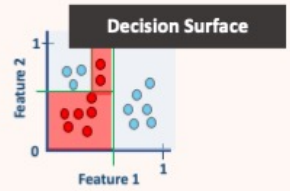


Terminal nodes (leaves), of the tree represents a cell of the partition, and has attached to it a simple model which applies in that cell only

- Example -

CART algorithm

- At each step, all predictors are tried and the best one (the one selected for splitting) is the one that maximizes the decrease in partition impurity (or some other metric).
- Then you take your child nodes and you split them again.
- and you iterate, until all nodes are leafs, or you meet some arbitrary stopping criteria like, max_depth, min nr of examples per split etc...
- You can use the same feature many times, but not with all DT alg.



PROS & CONS

PROS

- **Simple to understand and to interpret** - A white box model
- **can be visualized.**
- **for regression and classification problems**
- **Can generate complicated decision surfaces,** with little data
- **little data preparation.**
- DT can handle **numerical and categorical data**.
- It is possible to validate a model using **statistical tests** - It allows to account for the reliability of the model.
- **Performs well even if its assumptions are somewhat violated** by the true model from which the data were generated.
- Can work on feature spaces that are not smooth , and not continuous
- DT can be used to predict multi-class labels e. target var. is 2D array

CONS

- DTs may easily overfit the model to train data
- High Variance = DT can be unstable
- not good for extrapolation
- No detereministic solution
- Not all rules are easy to learn eg. XOR
- Big problems with class imbalance

IMPURITY MEASUREMENTS

ENTROPY & INFORMAITON GAIN

USED FOR: **ID3 alg**

- **values = [0, 1], unlike gini**
- **FAVORS smaller partitions with distinct values.**
- More difficult to calculate then gini
- *Used to calculate* **information Gain**
- A constant quantity has zero entropy - ie. its distribution is perfectly known. 0 == GOOD
- A uniformly distributed random variable (discretely or continuously uniform) maximizes entropy 1 == BAD

GINI IMPURITY

USED FOR **CART alg – sklearn !**

- **Values = [0, 0.5]**
- **FAVORS larger partitions, with easier tio implement splits**
- It performs only **Binary splits**. - works with the **categorical target variable** "Success" or "Failure".
- Higher value of Gini index implies higher inequality, higher heterogeneity. Information gain is a decrease in entropy.

GAIN RATIO

*USED FOR : **C4.5 – improvement of IDC3***

- ***modification of information gain***
- ***modification of information gain***
- **BIASED TOWARD selecting attributes with large number of values or nodes**
- **Takes into the account the nr of branches**
- Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split.
- It corrects information gain by taking the intrinsic information of a split into account.

IMPORTANT ISSUES

- **Stopping Criteria;** Eg. max depth, min number of samples in a leaf, min decrease in impurity. Some of these criteria are connected with the number of samples used in a training set.
- **Pruning:** you may remove tree nodes and leaves that do not meet your stopping criteria
- **Cross-Validation;** the analysis is conducted iteratively on random subsets of the data set, or validation of the resulting tree against a second, independent data set

MISSING DATA

- If present Sklearn return nan for predictions
- Use imputation, or remove them

SAMPLE NR

- *DT may overfit on data with a large number of features*
- *The number of samples must increase with the number of DT levels ~ double per each level*
- *Use: (i) feature selection or (ii) dimensionality red. (PCA, ICA)*

IMBALANCED DATASET

- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant
- *Use class weight*

ALGORITHMS

ID3 (Iterative Dichotomiser 3)

- The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets.
- Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalize to unseen data.

C4.5 is the successor to ID3

- removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals.
- C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules.
- These accuracy of each rule is then evaluated to determine the order in which they should be applied.
- Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

C5.0

- proprietary license.
- It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees)

- similar to C4.5
- supports numerical target variables (regression)
- does not compute rule sets as C4.5.
- constructs binary trees using the feature and threshold that yield the largest information gain at each node.
- **scikit-learn uses an optimized version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now.**

SOURCES:

- Lectures on DT: https://www.stat.cmu.edu/~cshalizi/350/lectures/22/lecture-22.pdf
- Splits with the same features: https://stats.stackexchange.com/questions/41105/can-two-or-more-splits-in-a-binary-decision-tree-be-made-on-the-same-variable
- **Plotting decision surface with pyhton:** https://www.analyticsvidhya.com/blog/2020/08/plotting-decision-surface-for-classification-machine-learning-algorithms/
- https://epurdom.github.io/Stat131A/book/regression-and-classification-trees.html

by Pawel Rosikiewicz

CLASSYFICATION TREES – PRACTICAL TIPS

www.SimpleAI.ch

STEP 1. ENSURE PROPER SAMPLE NR

ENSURE PROPER SAMPLE NR FOR FEATURE NUMBER

Decision trees tend to overfit on data with a large number of features, and small sample nr

- Getting the **right ratio of samples to number of features i**s important, since a tree with few samples in high dimensional space is very likely to overfit.
- **Consider performing dimensionality reduction** (PCA, ICA, or Feature selection) beforehand to give your tree a better chance of finding features that are discriminative.

HAVE ENOUGHT SAMPLES FOR A TREE DEPTH

The number os samples must increase with the number of DT levels

RECOMMENDATIONS

- **the number of samples required to populate the tree doubles for each additional level** the tree grows to.

HOW TO

- Use **max_depth** parameter to control the size of the tree to prevent overfitting
- Use **min_samples_split or min_samples_leaf** to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered.
    - **A very small number** will usually mean the tree will **overfit**, whereas a large number will prevent the tree from learning the data
    - Try min_samples_leaf=5 as an initial value.
    - **If the sample size varies greatly**, a float number can be used as percentage in these two parameters.
    - While **min_samples_split** can create arbitrarily small leaves, **min_samples_leaf guarantees that each leaf has a minimum size, avoiding low-variance**, over-fit leaf nodes in regression problems.
    - **For classification with few classes**, **min_samples_leaf=1** is often the best choice.
- **When dealing with imbalanced datasets:**
    - Note that **min_samples_split** considers samples directly and independent of sample_weight, if provided (e.g. a node with m weighted samples is still treated as having exactly m samples).
    - Consider **min_weight_fraction_leaf** or **min_impurity_decrease** if accounting for sample weights is required at splits.

STEP 2. PREPARE THE DATA

CLASS IMBALANCE

DT model may be strongly biased toward dominant class in training dataset

RECOMMENDATIONS

- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant - VERY IMPORTANT

HOW TO

- **Class balancing** – undersamplig, or sampling an equal number of samples from each class,
- Preferably - **normalizing the sum of the sample weights** (sample_weight) for each class to the same value.
    - **weight-based pre-pruning criteria,** such as **min_weight_fraction_leaf,** will then be **less biased toward dominant classes** than criteria that are not aware of the sample weights, like **min_samples_leaf.**
    - If the samples are weighted, it will be easier to optimize the tree structure using weight-based pre-pruning criterion such as min_weight_fraction_leaf, which ensure that leaf nodes contain at least a fraction of the overall sum of the sample weights.

FEATURE ENCODING & SPARSE DATASETS

Issue1: **All decision trees use np.float32 arrays internally.**

- If training data is not in this format, a copy of the dataset will be made.

Issue 2: **Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.**

- Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.

MISSING DATA

- If present Sklearn return nan for predictions
- Use imputation, or remove them

What to do next MODEL EVALUATION

Understanding the decision tree structure will help in gaining more insights about how the decision tree makes predictions, which is important for understanding the important features in the data.

RECOMMENDATIONS

Check the following when evaluating your model

- the depth of each node and whether or not it's a leaf;
- the nodes that were reached by a sample using the decision_path method;
- the leaf that was reached by a sample using the apply method;
- the rules that were used to predict a sample;
- the decision path shared by a group of samples.

HOW TO

- EXAMPLE with CODE: https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#sphx-glr-auto-examples-tree-plot-unveil-tree-structure-py
- Visualize your tree as you are training by using the export function. Use max_depth=3 as an initial tree depth to get a feel for how the tree is fitting to your data, and then increase the depth.
- retrieve the decision path of samples of interest

Intro to DT Classifier @ Kaggle https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial/notebook

Linear regression

- How to -

CHECKLIST USED FOR EDA & DEVELOPING LINEAR REGRESSION MODELS WITH NEW DATA

- Step 1 - DATA INGESTION

- Load & merge data into one dataframe with all features and representative number of examples
- Check df shape, indexes, duplicated cols & rows, and if you created new rows/columns with NA's

Rows/Col with too many NA

Optionally, Remove/or label samples/rows & columns with too many NA, or only NA

Anomaly Detection

Do you see any samples with more then average number of zeros/NAs/anomalies/novelties

- Step 2 - DATASET SUMMARY

Dataset Overview
- What is in my data?
- How many variables/samples?
- What are feature dtypes, and are they correctly encoded?

Missing Data
- How much of NA do you have in the dataset - per row/col, and eg: how many rows/col have >=90% of missing data
- What is the detailed distribution of missing data ie. which rows/columns
- What other values can be interpreted as NA, eg: 0, or "no value" text
- what NA, can be encoded as 0, or class/category

Feature Values
- what is the range or class number in each feature?
- Which var. have imbalanced class/value distribution,
- Get examples of each feature
- Are there any obvious OUTLIERS?
- How many zero values are there,

Check for spacial/datetime variables
- Check datetime features
- Identify columns with ID, sample order, or similar, and check if they are all unique

- Step 3 - DATA CLEANING

1. Define Variable Groups

(target, quantitative, ordinal, nominal, datetime, spatial, ID columns)

2 Cleaning & Organisation

Format dtypes
- Set proper dtype for each feature
- Make sure date-time variables are properly formatted,
- If exported from Excel (at any point) check for numerical values turned into data-time

Remove Duplicates
- Identify and remove any duplicates (rows & columns)

Missing data
- Check if all missing data are encoded in the same way, if not use one notation,,
- No imputation at this stage !
- The goals, is to have all missing data encoded as NA, and not something else or to remove, rows/columns that have only missing data

Remove or transform outliers,

- Step 4 - ENCODING

Define Encoding policy

The goal is to have consistent, and non-ambiguous encoding in entire dataset

For quantitative variables
- Identify incorrect values: Eg: -100 foot size, (plots, histograms, get examples)
- Next, prepare dict./note/script with (i) feature name, (ii) incorrect values, and (iii) actions to take to either Remove, Replace or clip identified incorrect values.

For ordinal variables
- Define how many classes, and how should they be called
- To help, checking for correlations, replace class names with sorted integers, means or medians (target var), and store it in a new column

For nominal Variables
- text/object; Tokenize text/object values or replace with classes
- Define proper encoding for each class Eg: remove inconsistent class names such as "House"/"Home" -> "Home"

For datetime Variables
- Make sure date-time variables are properly formatted, (one timezone)
- Encode these variables, eg: age, years/days since last renovation, etc...

- Step 5 - CHECK MODEL ASSUMPTIONS

TIP: transform nominal var's into ordinal using the target var. to include them in the analysis

Check Target Variable
- Does it have Normal distribution?
- Use: histogram, boxlot, QQ-plot + stat tests + skewness, kurtosis, mean, median
- Check if scaling, or log-transformation should be used to improve residuals disrib.

Perform Correlation analysis

- Pair-wise comparisons -
- First. Focus on Xi~dependent variable , to identify predictors correlated with the target variable (use: barplot, table)
- Then, analyse correlations between each variable pair, to find collinear var's that should be removed (use: annotated heatmap & table with sorted results)

- one vs rest comparison -
- Calculate ViF score for each feature, to detect potential muticolinearity (use: barplot, & table to present the results)

Check Quantitative variables

- Distribution -
- Are they normally distributed?
- If not, try transformations for each var. eg: log, sqrt, polynomial. Etc... list propositions,

- Extreme Values -
- Can you find outliers or groups of outliers/zeros in the data? (eg. using z-scores, on scatter plots)
- Do you see any samples with more then average number of zeros, Nas, anomalies, or novelties

Check linear model validity
- Main question: can we fot linear regression model with our data?
- Plot target variable against any predictor. Use, scatter plot with smoothed trend-line, or boxplots. Add info on correlation, NA%, mean, median, sd, to each plot,

- Step 6 - FEATURE ENGINEERING & F. SELECTION

GOAL: To generate 2D matrix with only numerical data (int, floats, binary). The selected, & transformed features should have following characteristics:

(i) No autococelation/batch effects
(ii) linear relationship with the target variable
(iii) multivariate normal distribution
(iv) Scaled or normalized values
(v) No missing data
(vi) Best, No outliers, or identified outliers, + vector with weight

use Sklearn functions to create transformers, and add them to one piepline

DEFINE
- Features that you wish to use in the model (see feature selection part in Notebook 2)
- transformers for each variable
- imputation strategy
- variables for one-hot encoding
- numerical variables that wiii be scaled, standardized of normalized, or other...

CHECK
- How does your transformers treat novelties, especially for nominal variables,

REMEMBER TO SAFE
- Feature names, and sample ID for each cell/row/column
- Dataset & pipeline version
- Source data

See Notebook 2

- Step 7 - MODEL TRAINING & EVALUTATION

Here we only establish a simple pipeline for testing assumptions and setting up automated functions

See Notebook 2

- Step 8 - ERROR ANALYSIS

1. Model performance assessment
- Bar-plot, or boxplot with errors calculated for the model/s and baseline (***RMSE, MAE, R^2***) to select best performing models
- Add test a& training error
- Print table with models sorted by test error, with information on hyperparameters and data transformations and features

2. Residual analysis

- Done for selected models -

Create Four Standard Plots
- **Plot 1. Histogram**
- **Plot 2. QQ-Plots or PP-Plots**
- **Plot 3. residuals vs target variable**
- **Plot 4. Residuals vs sample order/ID**

Test Residuals Normality
- Use plot 1 & 2
- Mean; Median, Skewness; Kurtosis;
- Shapiro–Wilk test, Scipy normaltest, Lilliefors or Kolmogorov-Smirnov

Test Residuals Homoscedascity
- Use plot 3 - fannning effect
- White test; or Breusch-Pagan test

Test for residuals autocorelation
- Use plot 4 – look for patterns
- Durbin-Watson Test

Step 3. Check for latent variables
- check if residuals/ target values are correlated with (i) amount of missing data/zeros per sample, (ii) sample order (plot 4) (iii) or the % of extreme values in each sample
- + Look for batch effects & autocorrelation,
- Then, use PCA or t-SNE to check if you can cluster the samples with any spatial var's

Additional Analyses:
- ***Model Resilance;*** *how much of missing data, novelties, or values outside feature space the model can tolerate*
- **Feature importance criterion**
- **Results stratification**
- ***Validation Dataset*** *-. used to test the final model*

by **Pawel Rosikiewicz**

ASSUMPTIONS BEHIND LINEAR REGRESSION

SimpleAI.ch

VALIDITY OF LR MODELS & LINEARITY

DESCRIPTION

X ~Y should be a linear relationship

- *(good example: Anscombe Quartet*

and

y is a linear combination of the parm's (coefficients) and predictor var's (X).

Consequences: Xi are treated as fixed points, thus, only coeff's must be found + you can create many copies/transforms of X, as long as they are not exactly corr. with each other

If Problematic:

Linear model can not be used or it will make systematic, and biased results

TESTING

Examine X~Y relationship

X~Y - IS IT LINEAR RELATIONSHIP?

- **Target variable vs predictor variable** ideally these should be linear

- Methods -

- *Scatterplots with smoothed line,*
- *Eg: sns.*
- *Boxplots, violin plots, with trendline*

X~Y - ARE THEY CORRELATED?

- Calculate Correlation coeff for each predictor vs target var.

- Methods -

- *Pearson (linear), Sperman (rank), or kendal corr. Coef (rank pairs)*
- *Barplot with sorted corr. results,*
- *Table,*

Check Input data

ARE ALL FEATURES NORMALLY DISTR.?

- **Plots**; *histogram, boxplot, and QQ-plot*
- ***Descriptive;*** *kurtosis, skewness, mean*
- **Stats;** *most often: Shapiro–Wilk , Kolmogorov-Smirnov (>300 samples)*

HANDLING PROBLEMS

Turn curved line of x~y into straight one &/or ensure that X values are norm. distr.

FEATURE TRANSFORMATIONS

Log, sqrt, power etc...
Not necessarily, to get norm distrib. values

ENSURE NORMALITY OF INPUT DATA

Especially important with >2 predictors

Fit model to a curved line of x~y

POLYNOMIAL REGRESSION

NON-LINEAR REGRESSION

NO MULTICOLINEARITY

DESCRIPTION

Muticolinearity: two or more independent variables have a high correlation among themselves.

If Problematic:

- Computational problems: have difficulty in distinguishing between effects of correlated predictors on the dependent variable.
- ADD MORE DATA – it often helps

TESTING

Examine Feature Values

FIND WHICH PAIRS OF PREDICTOR VARIABLES ARE CORRELATED WITH EACH OTHER

- ***Pair-wise comparisons (X)***

- Methods -

- ***Pearson** (linear), **Sp ....***
- ***Heatmap**, or table with top results (use absolute values, or head/tail)*

FIND WHICH PREDICTOR VARIABLE IS MOST LIKERY CORRELATED WITH THE REST OF THE DATA

- ***One-vs-rest comparisons (X)***

- Methods -

- **Variance Inflation Factor (VIF)** *one feature is used as dependent var. and all other features as its predictors*

HANDLING PROBLEMS

Remove one column in each one-hot-encoded feature

Regularization

RIGDE /ELASTICNET or LARS REGR

Remove corr. features

REMOVE HIGHLY CORR. FEATURES *(LASSO, VIF>5, corr~-1/1)*

HIERARCHICAL CLUSTERING

Compute spearman rank order coeff. and pick a single feature from each cluster of features correlated with each other

Dimensionality Reduction

PCA PREPROCESSING

take the top eigenvectors that preserve the max. variance

NORMALLY DISTRIBUTED RESIDUALS

DESCRIPTION

Required for validity of confidence intervals & model predictions

- In ideal situation, you would collect many Xi for each Y, and ideally, the measuring error that causes differences between Xi, would have normal distribution with mean=0.
- In most datasets, you don't have many Xi points for each Yi, but you can generalize, this assumption for all residuals calculated from entire Xi/Yi population.

TESTING

Are residuals v. normally distributed?

VISUAL INSPECTION OF RESIDUALS

Part 1

- ***Histogram***
- ***QQ-Plots** (Quantilie-Quantile Plots), or **PP-Plots** (Probability Plots)*
- ***Plot residuals vs target variable** (look for bias above/below 0)*

DESCRITIVE STATISTICS

- ***Mean;** should be 0*
- ***Skewness;** should be 0*
- ***Kurtosis;** ideally ~2.2*

STATISTICAL TEST FOR NORMALITY

- ***Shapiro–Wilk test** (small and medium size datasets)*
- **Scipy normaltest** (up to~50 samples);
- **Lilliefors-test** (50-300 samples),
- ***Kolmogorov-Smirnov** (>300 samples)*

HANDLING PROBLEMS

Try it in this order:

CHECK INPUT DATA NORMALITY

Use the same methods as in the above

IF NOT TRANSFORM THE DATA

Log, sqrt, repricotal, power etc...

TEST LINEARITY ASSUMPT. & ACT IF IT IS NOT HOLDING

See actions for the first assumption

APPLY NON-LINEAR TRANFORMERS TO IMPUT DATA

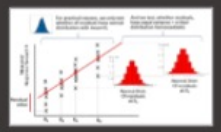
- Quantile transformer
- Power transformer
  - 'box-cox' – positive data only
  - 'Yeo-Johnson' - ± data

HOMOSCEDASITY OF RESIDUALS

DESCRIPTION

Homogeneity in variance of residuals = Equal Variance of Errors

often in time series data



If Problematic:

- regression prediction remains unbiased and consistent but inefficient - distinguishable variances are averaged to get a single variance that is inaccurately representing all the variances of the line (eg. small values have the same variance as large ones)

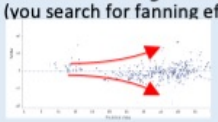
TESTING

Are residuals values homoscedastic?

VISUAL INSPECTION OF RESIDUALS

Part 2

- ***Plot residuals vs target variable*** (you search for fanning effect)



STATISTICAL TEST FOR HOMOSCEDACITY

- **White test**; detects, linear hetero-scedastity *Looses sensitivity, with large number of predictor variables*
- **Breusch-Pagan test** (more general, less sensitive)

Both tests works by creating an auxiliary regression, on the residuals, vs independent and dependent variables

HANDLING PROBLEMS

Y transformation

TRANSFORM DEPENEDENT VARIABLE

Log, sqrt, box-cox transformed, or power transformer , caution box-cox transf. changes the target value meaning

Modify or use different cost function

ADD WEIGHTS TO ERROR TERM

Weighted LS (WLS)

MODIFY ERROR FUCNTION

Eg. Huber loss, for dealing with outliers

NO AUTO-CORRELATION

DESCRIPTION

The correlation of a time series with its own past or the future values causes autocorrelation.

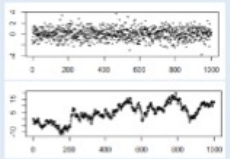
autocorrelation does not bias the OLS coefficient estimates. However, the standard errors tend to be underestimated

TESTING

Are residuals randomly distributed?

Plot Residuals vs

Sample Order, Time variable, any spatial variable, or any other variable, used in the model



No autocorelation *random assortment of residuals*

Autocorelation on the plot

DURBIN -WHATSON TEST

- measures the amount of autocorrelation in residuals from the regression analysis.
- check sONLY for the first-order autocorrelation, ie. lag =1

HANDLING PROBLEMS

IMPROVE THE MODEL

- Add new spatial/timedependent variables
- Tune hyperparameters
- Use Autoregressive models (AR1 model)

FEATURE TRANNSFORMATION

Transforming var's & test if the autocorrelations were reduced.

- deviation from the average values; eg: weather data.
- Log or exponential - may or may not make improvements.
- Annualising - to remove seasonal eff

CLUESTERING

Clustering on t time invariant factors.

BOX: ALL LINEAR REGRESSION ASSUMPTIONS

- *Validity of Linear Models*
- *No multicolinearity in the data*
- *Normally distributed residuals*
- *Homoscedastic residuals / constant variance*
- *No auto-correlation*

WHICH ASSUMPTION IS THE MOST IMPORTANT

First, check for validity of linear models, next independence assumptions, such as multicollinearity, and autocorrelation. Subsequently, test any equal variance assumption, and finally the assumptions on distribution (e.g., normal) - Techniques are usually least robust to departures from independence and most robust to departures from normality

Robustness to departures from normality is related to the Central Limit Theorem, since most estimators are linear combinations of the observations, and hence approximately normal if the number of observations is large

If a linear model fits with all predictors included, it is *not* true that a linear model will still fit when some predictors are dropped. For example, if $E(Y|X_1, X_2) = 1 + 2X_1 + 3X_2$ (so that a linear model fits when Y is regressed on both X_1 and X_2), but $E(X_1|X_2) = \log(X_1)$, then it can be calculated that $E(Y|X_1) = 1 + 2X_1 + 3\log(X_1)$, which says that a linear model does not fit when y is regressed on X_1 alone.

VALIDITY OF LR MODELS & LINEARITY ASSUMPTION

linearity assumption *- the conditional means of the response variable are a linear function of the predictor variable. Graphing the response variable vs the predictor can often give a good idea of whether or not this is true. However, one or both of the following refinements may be needed:*

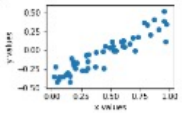
HOW TO TEST IT?

➢ VISUAL INSPECTION of x~y & Correlation analysis

CAUTION:
It is not possible to gauge from scatterplots whether a linear model in more than two predictors is suitable.

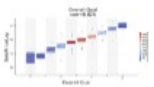
Plot residuals (instead of response) vs. predictor.

- A non-random pattern suggests that a simple linear model is not appropriate; you may need to transform the response or predictor, or add a quadratic or higher term to the mode.



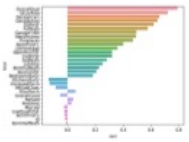
Use a scatterplot smoother

- **Eg: lowess** (also known as loess)
- It gives give a visual estimation of the conditional mean. Such smoothers are available in many regression software packages.
- *Caution*: You may need to choose a value of a smoothness parameter. Making it too large will oversmooth; making it too small will not smooth enough



Correlation analysis

- *Eg: Use Pearson's correlation, but, remember to Check coefficient value and the significance of the correlation (p-value) - p-value is indicating that there is an absence or presence of a Significant relationship between variables*
- *Example: barplot with sorted correlation values between each variable and the target variable*



HANDLING

FIRST:
We try to turn curved line of x~y relationship into straight one, and to apply linear regression model s – these are easier

➢ SOLUTION 1. TRY FEATURE TRANSFORMATIONS

- **Scaling & shift (f(x)±c)** – it helps more with model optimisation
- **Reflection** (-f(x), shifts values on the other site of the axis.
- **log transformation** (log(x), ln(x), used for right-skewed distributions),
  - Data skewed to the right (i.e. in the positive direction).
- **Sqrt transformations;** it basically makes a straight line from power function line, and its used to compresses larger values + it makes differences between small values more apparent. Less aggress then log transf.
- **power transformation** (eg: x^2), used when The data's relationship is close to an exponential model
- **repricotal transformation** (1/x, - dramatic effect on the shape of the distribution, reversing the order of values with the same sign. The transformation can only be used for non-zero values !),
- **Share mapping** (all points along one line stay fixed, while other points are shifted parallel to the line by a distance proportional to their perpendicular distance from the line)
- More info from https://www.calculushowto.com/transformations/

➢ THUS, YOUR ULTIMATE GOAL IS TO TRANSFORM PREDICTORS TO APPROX. MUTIVARIATE NORMALITY

Ii will ensure not only that a linear model is appropriate for all (transformed) predictors together, but that a linear model is appropriate even when some transformed predictors are dropped from the model.

BOX: multivariate normality

- multivariate normal distribution, multivariate Gaussian distribution, or joint normal distribution
- a generalization of the one-dimensional (univariate) normal distribution to higher dimensions.
- Def:is a random vector is said to be k-variate normally distributed if every linear combination of its k components has a univariate normal distribution.

Bivariate normal joint density

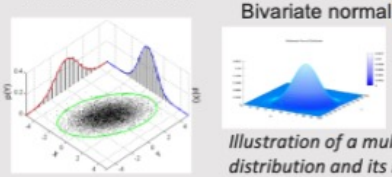


Illustration of a multivariate gaussian distribution and its marginals.

THEN:
If solution 1 fails, we must fit the model to curved x~y line

➢ SOLUTION 2. USE POLYNOMIAL REGRESSION

Allow for the polynomial linear regression equation. While the independent variable is raised to power of 2 here, the model is still linear in terms of its parameters. Linear models can also contain log terms and inverse terms to follow different kinds of curves and yet continue to be linear in the parameters

➢ SOLUTION 3. APPLY NON-LINEAR REGRESSION MODEL

- Regression trees
- Nonlinear regression uses logarithmic functions, trigonometric functions, exponential functions, power functions, Lorenz curves, Gaussian functions, and other fitting methods

NO MULTICOLINEARITY IN INPUT DATA

NEAR COLINEARITY & ILL-CONDITIONING

Nearly Col. F's, have "imperfect" or close to linear relationship

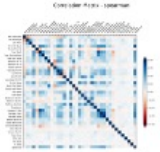
- *Eg: weight from 2 diff. balances*
- **OLS solution exist**, but the calculations are unstable
- **Ill-Conditioning:** matrix with nearly colinear features, has large condition nr that is used to find OLS solution, and it is susceptible to small changes in input data. Thus it causes large variation in model coeff. with small changes in the data
- In most cases, ill-conditioning doesn't affects model accuracy!

COLINEAR FEATURES

exact linear relationship among two or more independent variables

- *Eg: weight in kg. and pounds*
- **Collinearity** causes problems in numerical computations. Eg: alg tries to decide which of the same variables are more important
- No solution for **OLS method**, == all features must be linearly independent.
- **Most alg.** can cope with that, but they will return warnings
- Collinearity is can be introduced by poor design & must be detected / handled before building a model

HOW TO DETECT MUTICOLINEARITY ?



$$VIF_i = \frac{1}{1 - R_i^2}$$

If R^2 ->1*, then the regr. Valuesin the i-th feature can be easily predicted with the rest of the data == potential mucticolinearity,* ***(VIF >>1)***

If R^2 -> 0*, then LR model is "bad" and there is proof for collinearity between i-th feature and the rest of the dataset* ***(VIF ->1)***

➢ Correlation analysis

- pair-wise comparisons only

➢ Variance Inflation Factor (ViF)

- Allows estimating multicollinearity in entire dataset, vs each feature,
- The value of VIF is computed for each feature, where a regression model is trained keeping one feature as dependent var. and all other features as its predictors

Values = [0, inf)

- VIF=1: No multicollinearity
- VIF 1 - 5: Moderate multicollinearity
- VIF > 5: Highly multicollinear

HANDLING

➢ SOLUTION 1. REMOVE HIGHLY CORR. FEATURES

- LASSO REGRESSION
- REMOVE HIGHLY CORRELATED FEATURES
  foe example: With Vif >= 5 /or/. Corr ~-1 or 1

➢ SOLUTION 2. USE RIGDE /ELASTICNET or LARS REGR

Ridge — used for data with large # of predictors of about the same value

ElasticNet — Lasso is likely to pick one of correlated features at random, while elastic-net is likely to pick both, but with smaller coefficients,

LARS — alternative to ridge & ElasticNet for multiple colinear features, iterative approach, **More effective for p>>n datasets, and alfa exploration**

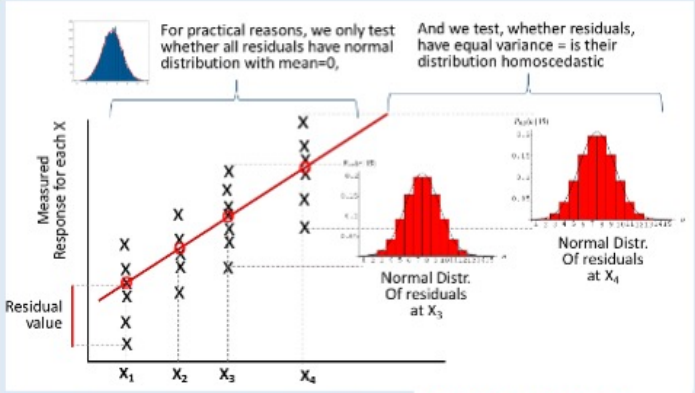
➢ SOLUTION 3. DIMENSIONALITY REDUCTION : PCA PREPROCESSING

taking the top eigenvectors that preserve the maximum variance. The number of dimensions can be decided by observing the variance preserved for each eigenvector.
https://github.com/bhattbhavesh91/pca-multicollinearity/blob/master/multi-collinearity-pca-notebook.ipynb

➢ SOLUTION 4. HIERARCHICAL CLUSTERING

Perform hierarchical clustering on the spearman rank order coefficient and pick a single feature from each cluster based on a threshold. The threshold can be decided by observing the dendrogram plots.

NORMAL DISRT. OF RESIDUALS/ERRORS



EFFECT: Heteroscedasticity will result in the averaging over of distinguishable variances around the points to get a single variance that is inaccurately representing all the variances of the line. In effect, residuals appear clustered and spread apart on their predicted plots for larger and smaller values for points along the linear regression line, and the mean squared error for the model will be wrong.

CAUTION: Hypothesis tests for equality of variance are often not reliable, since they also have model assumptions and are typically not robust to departures from these assumptions.

Ideally the residuals are:
- *normally distributed*
- *homoscedastic*
- *Independent from each other and form predicted value*

Independence is defined in two ways
1. Independence of errors from the values of the independent variables.
2. Independence of the independent variables.

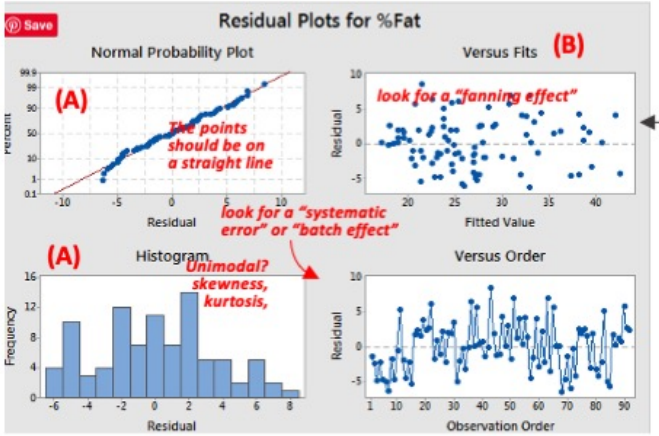
- BOX -
WHY IT IS IMPORTANT TO KNOW THESE ASSUMPTIONS

Linear Models are used most effectively with data, residuals, and target variables that adhere to assumptions of the linear regr. Thus, our goal in EDA, error analysis, and data transformation, is to explore whether each of these elements is following each assumption and take actions in case they are not.

HOW TO TEST IT?

➢ CHECK DISTRIBUTION OF RESIDUALS ON PLOTS

(A) Use: Quantilie-Quantile Plots (QQ-Plots), PP-Plots (Probability Plots) the goal is to check if the points from two distributions lie on the y~x line.



➢ Use Statistical tests for normality

Small sample-numbers (<50)
- **Shapiro–Wilk W test;** depends on the cov. matrix between the order statistics of the observations – less sensitive to outliers then normal test
- **scipy normaltest;** combines skew and kurtosis

Intermediate sample numbers (50-300)
- **Shapiro–Wilk W or Lilliefors-test**
- **Lilliefors-test;** based on Kolmogorov–Smirnov test, - quantifies distance between empirical distr. fn. of the sample and the cdf of the reference distrib (eg normal distrib), or between dictrib. Fn's of two samples. It is good because original K-S-test is unreliable when mean and std are unknown.

large sample numbers (>300)
- While having sufficient sample size, **Kolmogorov-Smirnov** and **Lilliefors-test**, are the least affected with extreme values (outliers), followed with **Shapiro–Wilk test**, that is more affected, but less then the **normaltest**

RESIDUALS HOMOSCEDATICITY

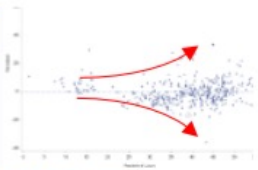
Homogeneity in variance of the residuals
= Equal Variance of Errors

***CONSEQUENCES**: The regression prediction remains unbiased and consistent but inefficient. It is inefficient because the estimators are no longer the Best Linear Unbiased Estimators (BLUE). The hypothesis tests (t-test and F-test) are no longer valid.*

HOW TO TEST IT?

➢ Plot Residuals vs Fitted Values (Plot B)

Look for **fanning effect** on the scatterplot with residuals vs the dependent variable



➢ Statistical tests Homoscedastic

- **White test:** *>>> statsmodels.stats.diagnostic.het_white*
- **Breusch-Pagan test** *>>> het_breuschpagan in statsmodels.stats.diagnostic*

SIMILARITY BETWEEN THESE TWO TESTS
- Both tests works, similar by creating an auxiliary regression, on the residuals, vs independent and dependent variables
- The errors are heteroscedastic when p>0.05

& DIFFERENCES
- The **Breusch-Pagan** test only checks for the linear form of heteroskedasticity
- The **White test** is more generic. It relies on the intuition that if there is no heteroskedasticity the classical error variance estimator should gives you standard error estimates close enough to those estimated by the robust estimator (based on median). A shortcoming of the White test is that it can lose its power very quickly particularly if the model has many regressors.

HANDLING

➢ SOLUTION 1. TRANSFORM DEPENEDENT VARIABLE

Typically we use log, transformation, or box-cox tranformations (power transformer for positive only, values) Comment: it often happens in time series data, caused by season, monthly, and other patterns

➢ SOLUTION 2. ADD WEIGHTS TO ERROR TERM -

eg use Weighted LS LR – errors are heteroscedastic, but still indepoendent from each oter (ie. no autocorrelation, if you. See that, look atb Generalized LR models, GLR)

use weighted least squares in case all the transformations fail to solve the problem at hand. LinearRegressor, in Sklear, takes weights as the third parameter after X, y, in sest. Fit() . the weights are multiplied by errors, calculated from each data point – hence scaling of weights doesn't change anything.

How to construct weights
1. Compute the absolute and squared residuals
2. Find the absolute and squared residuals vs. independent var's to get the estimated standard deviation and variance
3. Compute the weights using the estimated standard deviations & variance.

➢ SOLUTION 3. MODIFY ERROR FUCNTION

eg use Huber Regression

HANDLING

SOLUTION 1.
➢ TEST LINEARITY ASSUMPTION & ACT IF IT IS NOT ....

First check if the linearity assumption is being violated. That can cause a failure with the normality assumption as well.

SOLUTION 2.
➢ CHECK INPUT DATA NORMALITY & TRANSFORM THEM TO HAVE IT

Perform univariate analysis of dependent & independent variables and see **if the are significantly deviating from the normal distribution (see statistical tests for normality in the above)**. In this you can transform features (log, sqrt, power, or repricotal etc.)

SOLUTION 3.
➢ APPLY NON-LINEAR TRANFORMERS TO IMPUT DATA

- **Quantile transformer**
- **Power transformer**
  - **'box-cox'** - needs the data to be positive
  - **'Yeo-Johnson'** - data to be both negative and positive.

PART 1 SOURCES
Much of the text on this slide, and the figure taken from:
https://medium.com/geekculture/holy-grail-for-understanding-all-the-assumptions-of-linear-regression-210f224192b5
https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/
https://stackoverflow.com/questions/42658379/variance-inflation-factor-in-python
https://towardsdatascience.com/7-techniques-to-handle-multicollinearity-that-every-data-scientist-should-know-ffa03ba5d29
https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance_multicollinear.html#handling-multicollinear-features
https://web.ma.utexas.edu/users/mks/statmistakes/modelschecking.html
https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance_multicollinear.html
https://github.com/bhatthavesh91/pca-multicollinearity/blob/master/multi-collinearity-pca-notebook.ipynb

PART 2 SOURCES
https://realpython.com/numpy-scipy-pandas-correlation-python/#rank-correlation
https://docs.tibco.com/data-science/GUID-75A8887A-9927-4772-BC83-DA09E6583387.html
https://statisticsbyjim.com/regression/variance-inflation-factors/
https://medium.com/geekculture/holy-grail-for-understanding-all-the-assumptions-of-linear-regression-210f224192b5
https://stackoverflow.com/questions/42658379/variance-inflation-factor-in-python

NO AUTO-CORRELATION
- Independence of errors -

ABOUT AUTOCORRELATION

A measure of similarity between a given time series and the lagged version of the same time series over successive time periods. In other words: It is a correlation between two different versions X_t and X_{t-k} of the same time series.

Partial autocorrelation function PACF

- PACF gives the partial correlation of a stationary time series with its own lagged values, regressed the values of the time series at all shorter lags. It is different from the autocorrelation function, which does not control other lags then 1.

Causes

- The correlation of a time series with its own past or the future values causes autocorrelation. Generally, any usage has a tendency to remain in the same state from one observation to the next. This specific form of 'persistence' causes the positive autocorrelation.

Effects

- autocorrelation it does not bias the OLS coefficient estimates. However, the standard errors tend to be underestimated

Usage

- **For checking randomness in the time-series**; In many statistical processes, our assumption is that the data generated is random (autocorrelation of lag 1)
- **To determine whether there is a relation between past and future values** of time series, we try to lag between different values.

Challenge with testing independence assumptions

- Independence Assumption are usually formulated in terms of error terms rather than in terms of the outcome variables. For example, in simple linear regression, the model equation is $Y = \alpha + \beta x + \varepsilon$, where Y is the outcome (response) variable and ε denotes the error term (also a random variable).
- It is the error terms that are assumed to be independent, not the values of the response variable. We do not know the values of the error terms ε, so we can only plot the residuals e_i (defined as the observed value y_i minus the fitted value, according to the model), which approximate the error terms.

BOX: ALL LINEAR REGRESSION ASSUMPTIONS

- *Validity of Linear Models*
- *No multicolinearity in the data*
- *Normally distributed residuals*
- *Homoscedastic residuals / constant variance*
- *No auto-correlation*

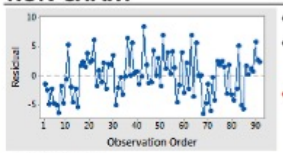
SOURCES
https://kumaradevan.com/how-to-handle-autocorrelation/
https://stats.stackexchange.com/questions/14914/how-to-test-the-autocorrelation-of-the-residuals
https://stats.stackexchange.com/questions/50151/how-to-tell-if-residuals-are-autocorrelated-from-a-graphic?noredirect=1&lq=1
https://www.geeksforgeeks.org/autocorrelation/

HOW TO TEST IT?

➢ CHECK RESIDUALS – VISUAL INSPECTION

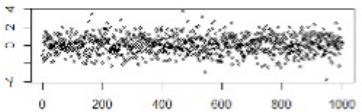
Plot Residuals vs
- ➢ Sample Order
- ➢ any Time variable
- ➢ any spatial variable
- ➢ any variable, used in the model

RUN CHART

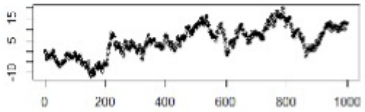


- ***Residuals vs sample order***
- *Line plot is ioten used – it helps visualizing the order of connected points points*
- *RESULTS INTEPRETATION: A pattern that is not random suggests lack of independence.*

EXAMPLE



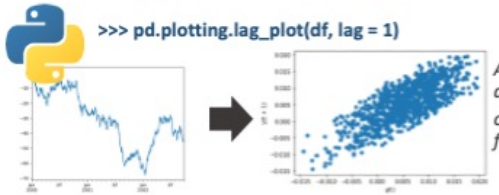
No autocorelation
random assortment of residuals



Autocorelation on the plot
If we compare each value to their preciding/subsequent values in order on X axis, high values will be often correlated with similarly high values, and low values with similarly low values, irrespectively on the pattern

LAG PLOT/ CORRELOGRAM

- a special type of scatter plot (but not a scatterplot !) with the two variables, showing X, and X-lagged. eg: One set of observations in a time series is plotted (lagged) against a second, later set of the same data.
- **A "lag"** is a fixed amount of passing time or sample order;
- **First-order lag plot.** – the plot with The **most commonly used lag == 1**.



Autocorelation plot can be done with pandas function, other plots are also available for deeper anaylsis

➢ STATISTICAL TESTS FOR AUTOCORELATION

DURBUIN–WATSON TEST

ABOUT
- Used to measure the amount of autocorrelation in residuals from the regression analysis.
- It is used to check ONLY for the first-order autocorrelation, ie. lag =1

ASSUMPTIONS
- The errors are normally distributed and the mean is 0.
- The errors are stationary.

FORMULA

$$DW = \frac{\sum_{t=2}^{T}(e_t - e_{t-1})^2}{\sum_{t=1}^{T} e_t^2}$$

e_t - residual of error from the Ordinary Least Squares (OLS) method

VALUES & RESULTS INTEPRETATION

The Durbin Watson test has values between 0 and 4.
- **2: No autocorrelation. Generally, we assume 1.5 to 2.5 as no correlation.**
- **0- <2: positive autocorrelation.** The more close it to 0, the more signs of positive autocorrelation.
- **>2 -4: negative autocorrelation.** The more close it to 4, the more signs of negative autocorrelation.

HANDLING

➢ IMPROVE THE MODEL

Try to capture structure in the data in the model. ...
- Adding other independent variables
- Experimenting with model specification
- Use ***Autoregressive model*** (AR1 model)

➢ FEATURE TRANNSFORMATION

- Transforming variables into different functional forms;
- Many variables can be transformed into different forms and tested to see if the autocorrelations were reduced. (log, exponential, annualized etc...)
- Examples:
  - deviation from the average values; eg: weather data.
  - Log form or exponential form may or may not make improvements.
  - Annualising the data - it is supposed to remove any seasonal effects.

➢ CLUSTERING

- **Clustering data on different time invariant factors**
- Clustering based on variables, such as income and lot size, can improve the autocorrelations problems. This may be due to the phenomenon of seasonality, with houses reacting differently to the same weather conditions.
- Caution. p values of estimates can get worse as the number of households within each cluster is smaller than the whole segment. Hence, the clustering analysis can be used to identify outliers and appropriate actions can be taken.

Clustered standard errors
https://en.wikipedia.org/wiki/Clustered_standard_errors

CAUTION

These test, have problems with detecting many types of autocorrelation, thus, visual inspection is always recommended + you can perform more plots to support your claims for autocorrelation at different lag values

by **Pawel Rosikiewicz**

Analysis of residuals – plots used

Graphical analysis of residuals

A basic, though not quantitatively precise, way to check for problems that render a model inadequate is to conduct a visual examination of the residuals (the mispredictions of the data used in quantifying the model) to look for obvious deviations from randomness. If a visual examination suggests, for example, the possible presence of heteroskedasticity (a relationship between the variance of the model errors and the size of an independent variable's observations), then statistical tests can be performed to confirm or reject this hunch; if it is confirmed, different modeling procedures are called for.

Different types of plots of the residuals from a fitted model provide information on the adequacy of different aspects of the model.

1. sufficiency of the functional part of the model: scatter plots of residuals versus predictors
2. non-constant variation across the data: scatter plots of residuals versus predictors; for data collected over time, also plots of residuals against time
3. drift in the errors (data collected over time): run charts of the response and errors versus time
4. independence of errors: lag plot
5. normality of errors: histogram and normal probability plot

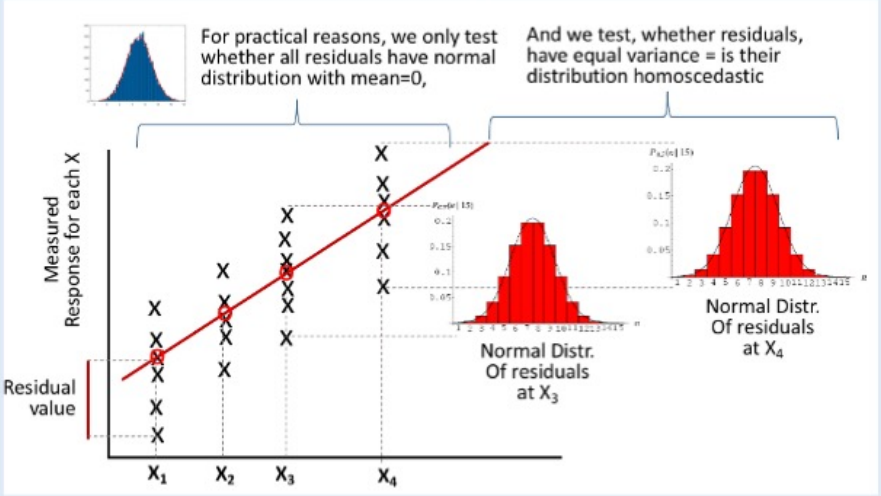
Graphical methods have an advantage over numerical methods for model validation because they readily illustrate a broad range of complex aspects of the relationship between the model and the data.

Ideally the residuals are:

- *normally distributed*
- *homoscedastic*
- *Independent from each other and predicted value*

Independence is defined in two ways

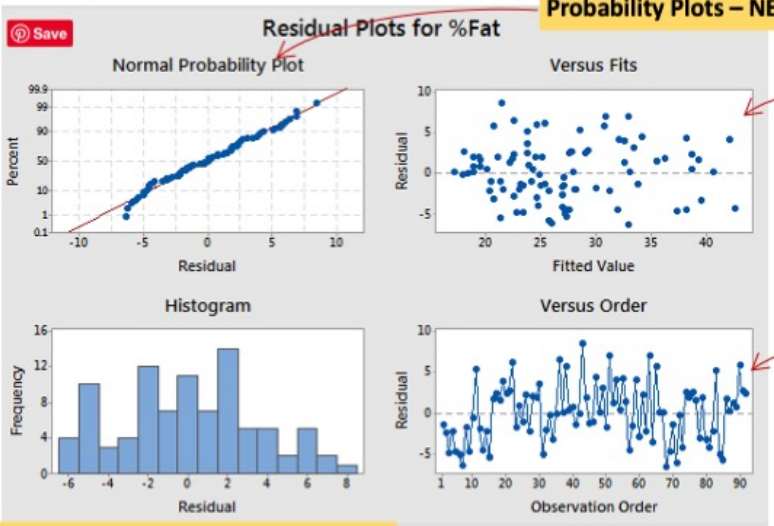
1.Independence of errors from the values of the independent variables.
2.Independence of the independent variables.



Plots typically used in analysis of residuals

Conclusion for our example

The residual plots below shows the unbiased fit because the data points fall randomly around zero and follow normal distribution



Probability Plots – NEXT PAGE

Comparison of Probability plots, histograms and boxplots on the next page

Scatter plot with residuals~fitted value

- the relation between fitted values (x.axis), and the residuals at these values.
- An ideal model, would produce random and homoscedastic residuals, with mean==0
- In case of the problems, we would see a "funnel effect" caused with larger residuals on larger predicted variables,

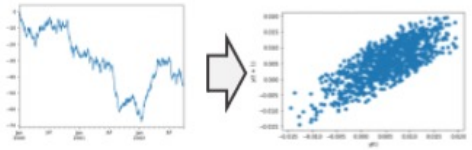
Run Chart

- A special type of plot, with time sequence or sample order on x-axis, and residual values on y-axis
- Typically, we use a line plot, with residual values connected in order to see the sample order on the plot directly,
- An ideal model would produce random ly and homosdedastically distributed residuals, with mean ==0

Lag Plot/Correlogram

- Not a scatterplot !
- Not used in our example,
- A lag plot is a special type of scatter plot with the two variables (X,Y) "lagged." A "lag" is a fixed amount of passing time; One set of observations in a time series is plotted (lagged) against a second, later set of data. ... The most commonly used lag is 1, called a first-order lag plot.
- It allows detecting is there is autocorrelation within the data, eg is sample 38, had lower residual then 37, and this then 36 etc..., creating a pattern. That allows predicting the next residual form the previous one.

```
>>> pd.plotting.lag_plot(df, lag = 1)
```



Autocorelation plot can be done with pandas function, other plots are also available for deeper anaylsis

SOME MATERIALS COPIED FROM, The figure on the top right is mine☺
https://en.wikipedia.org/wiki/Regression_validation

by Pawel Rosikiewicz

COREELARTION COEFFICIENTS

www.SimpleAI.ch

CORRELATION TYPES

There are several types of correlation metrics that can be used for different types of data: The most popular are

- **Pearson's coefficient** that measures linear correlation in numerical data
- **and Spearman or Kendall coefficients used to** compare the ranks of data

Correlation Technique	Relationship	Datatype of features
Pearson	Linear	Quantitative and Quantitative
Spearman	Non-linear	Ordinal and Ordinal
Point-biserial	Linear	Binary and Quantitative
Cramer's V	Non-linear	Categorical and Categorical
Kendall's tau	Non-linear	Two Categorical or Two Quantitative

CODE EXAMPLES

SciPy

All Scipy Functions Return
-> corr. coefficient
-> *p-value*

CAUTION
Check NA policy in each function

```
# create example data
>>> x = np.arange(10, 20)
>>> y = np.array([2, 1, 4, 5, 8, 12, 18, 25, 96, 48])

# Pearson's r
>>> scipy.stats.pearsonr(x, y)
(0.7586402890911869, 0.010964341301680832)

# Spearman's rho
>>> scipy.stats.spearmanr(x, y)
SpearmanrResult( correlation=0.975757, pvalue=1.4675461877e-06)

# Kendall's tau
>>> scipy.stats.kendalltau(x, y)
KendalltauResult(correlation=0.911111, pvalue=2.976190462e-05)
```

PEARSON R CORRELATION COEEFICIENT

The *correlation coefficient* between two variables answers the question: "Are the two variables related? That is, if one variable changes, does the other also change?" If the two variables are normally distributed, the standard measure of determining the correlation coefficient, often ascribed to Pearson, is

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \quad (11.1)$$

With the sample covariance s_{xy} defined as

$$s_{xy} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1} \quad (11.2)$$

and s_x, s_y the sample standard deviations of the x and y values, respectively, Eq. 11.1 can also be written as

$$r = \frac{s_{xy}}{s_x \cdot s_y}. \quad (11.3)$$

Pearson's correlation coefficient, sometimes also referred to as *population correlation coefficient* or *sample correlation*, can take any value from -1 to $+1$. Examples are given in Fig. 11.1. Note that the formula for the correlation coefficient is symmetrical between x and y—which is not the case for linear regression!

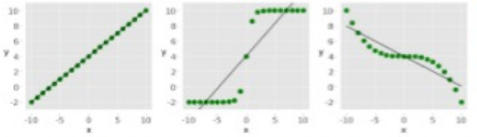
TWO RANK CORRELATION COEFFICIENTS

- ***For Not normally distributed data***
- ***For Non-linear relationships between compared var's***

➢ SPEARMAN RHO

The **Spearman correlation coefficient** Is calculated the same way as the Pearson correlation coefficient but takes into account their ranks instead of their values.

- Denoted with the Greek letter rho (ρ) and called **Spearman's rho**.
- $-1 \le \rho \le 1$.
  - If ρ=1 => the function between x and y increases monotonically
  - If ρ=1 => the function between x and y decreases monotonically



Rank correlation, allows comparing variables with non-linear relationships

Spearman Rho uses the ranks or the orderings of data points in compared variables/features, If the orderings are similar, then the correlation is strong, positive, and high. However, if the orderings are close to reversed, then the correlation is strong, negative, and low. In other words, rank correlation is concerned only with the order of values, not with the particular values from the dataset.

➢ KENDAL TAU

Greek letter tau (τ)

- **Kendal Tau is harder to calculate** than Spearman's but its confidence intervals are **more reliable**.
- calculated as the difference in the counts of concordant and discordant ranked data pairs, relative to the total number of x-y pairs in compared rankings.
- **VALUES**
  - **Always between $-1 \le \tau \le 1$**
  - **If $\tau = 1$** - the ranks of the corresponding values in **x** and **y** are the same. In other words, all pairs are concordant.
  - **If $\tau = -1$** - the rankings in **x** are the reverse of the rankings in **y**. In other words, all pairs are discordant.

HOW IT IS DONE

- Lets, have two random variables x, & y, that we wish to compare
- All values in x, and y were ranked independently.
- Now, if we compare all pairs of values in x, and y, using the same pairs in xi:xj, and yi:xj in both variables at each comparison, we will label the results in one the 3 following classes:
  - **Pair of points has concordant ranks in x,y;** *($x_i > x_j$ and $y_i > y_j$) or ($x_i < x_j$ and $y_i < y_j$)*
  - **… has discordant** ranks in x,& y ; *($x_i < x_j$ and $y_i > y_j$) or ($x_i > x_j$ and $y_i < y_j$)*
  - **… the same ranks in x&y (tie);** *a tie in x ($x_i = x_j$) or a tie in y ($y_i = y_j$), or in both,*

According to the `scipy.stats` official docs, the Kendall correlation coefficient is calculated as $\tau = (n^+ - n^-) / \sqrt{(n^+ + n^- + n^x)(n^+ + n^- + n^y)}$, where:

- n^+ is the number of concordant pairs
- n^- is the number of discordant pairs
- n^x is the number of ties only in **x**
- n^y is the number of ties only in **y**

If a tie occurs in both **x** and **y**, then it's not included in either n^x or n^y.

KENDAL TAU – DIFFERENT TYPES

- **scipy.stats.kendalltau has a.&b, variants,** that treat ties differently, b is default
- additionally, scipy offers **weighted kendal tau,** in which exchanges of high weight are more influential than exchanges of low weight.

REPORTING CORRELATION RESULTS

Reporting individual Results:

- … and … were found to be moderately positively corr., $r(38) = .34$, $p = .032$.
- …. were found to be strongly correlated, $r(128) = .89$, $p < .01$.
- …. were negatively correlated, $r(78) = -.45$, $p < .001$
- no linear correlation was found, between ..&…, $r(38) = .02$, $p = .005$

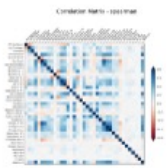
Comments

- Degrees of freedom for *r* is *N* - 2 – denoted in brackets -> r(120)
- Report the exact *p*value, + state your alpha, eg 0.05 level early in your results
- *r* statistic should be stated at 2, or 3 decimal places

CORRELATION MATRIX & TRENDLINE

Pandas

```
>>> df_corr = df.corr(method='pearson')
>>> fig, ax = plt.subplots()
>>> ax.matshow(
        df_corr, cmap=cmap, vmin= -1, vmax=1)
    # min,max values will scale the heatmap
```



Scipy

```
>>> rho, pvalues = scipy.stats.spearmanr(np.array)
```

Obtained with linear regr. In scipy

```
>>> result = scipy.stats.linregress(x, y)
    result.slope      # 7.4363636363636365
    result.intercept  # -85.92727272727274
    result.rvalue     # 0.7586402890911869
    result.pvalue     # 0.010964341301680825
    result.stderr     # 2.257878767543913
```

INTERPRETATION

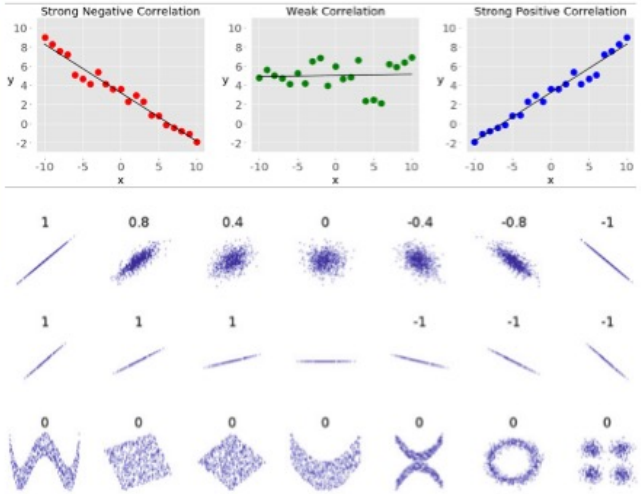


Fig. 11.1 Several sets of (x, y) points, with the correlation coefficient of x and y for each set. Note that the correlation reflects the noisiness and direction of a linear relationship (*top row*), but not the slope of that relationship (*middle*), nor many aspects of nonlinear relationships (*bottom*). N.B.: the figure in the center has a slope of 0 but in that case the correlation coefficient is undefined because the variance of Y is zero. (In Wikipedia. Retrieved May 27, 2015, from http://en.wikipedia.org/wiki/Correlation_and_dependence.)

SOURSES

https://stackoverflow.com/questions/42658379/variance-inflation-factor-in-python
https://towardsdatascience.com/7-techniques-to-handle-multicollinearity-that-every-data-scientist-should-know-ffa03ba5d29
https://realpython.com/numpy-scipy-pandas-correlation-python/#rank-correlation
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.weightedtau.html#scipy.stats.weightedtau

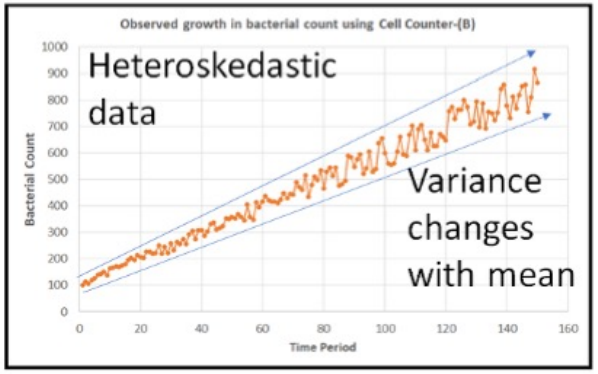
Examples of Variance Stabilizing Transformations

https://timeseriesreasoning.com/contents/generalized-linear-regression-models/

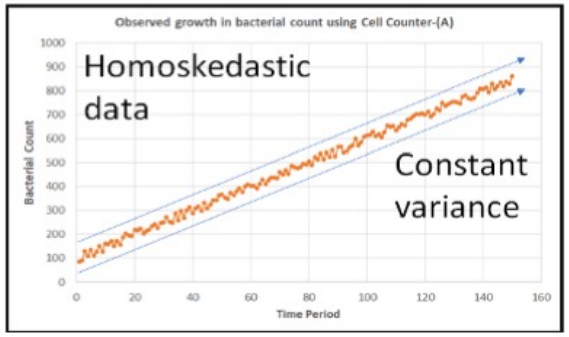
- Three major transformations of the dependent variable (Y) that may help correct the problem of heteroscedasticity:

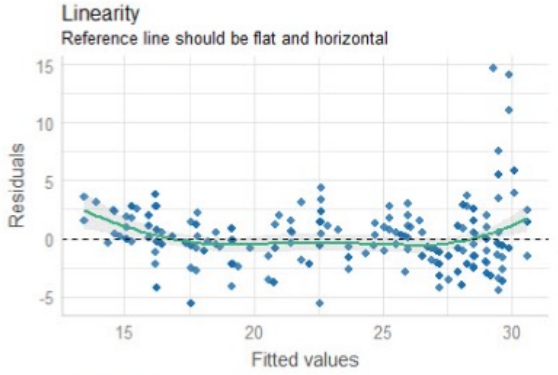
Type of Transformation	Structure	Explanation
Square root	$\sqrt{Y}$	Mildest transformation. Useful when the variance of the residuals is approximately proportional to the mean value of Y.
Logarithmic	ln(Y)	Moderate transformation. Here, we regress ln(Y) on the X variables. Reason is that log transformation compresses the scales in which the variables are measured, reducing a tenfold difference between two values to a twofold difference. Ex, 80 is 10x the number 8. But ln80 (=4.38) is only about 2x the ln8 (=2.08)
Reciprocal	1/Y	This is the most "severe" of all three transformations. Required when the violation of equal variance is serious. Useful when the variance of the errors is approximately proportional to the mean of Y to the 4th power.

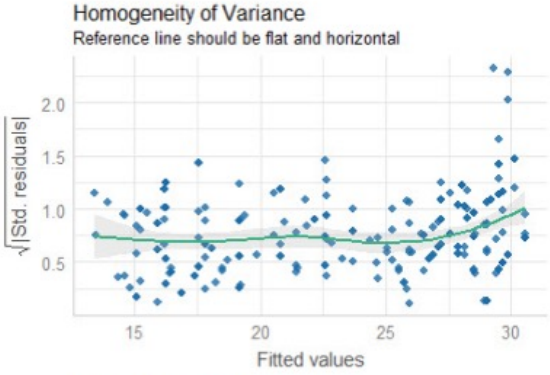
Most common

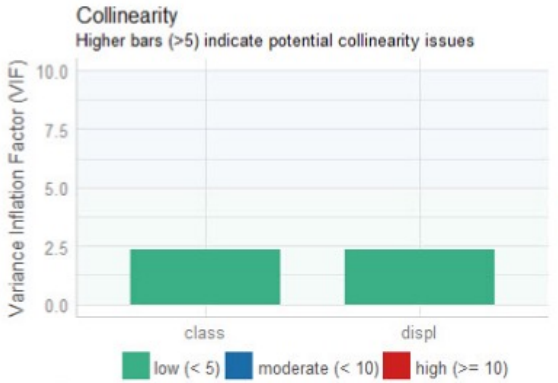


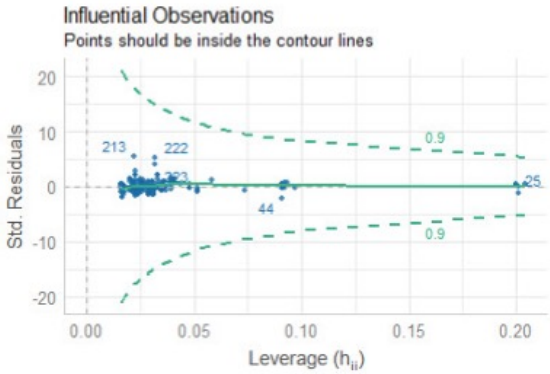
https://timeseriesreasoning.com/contents/generalized-linear-regression-models/

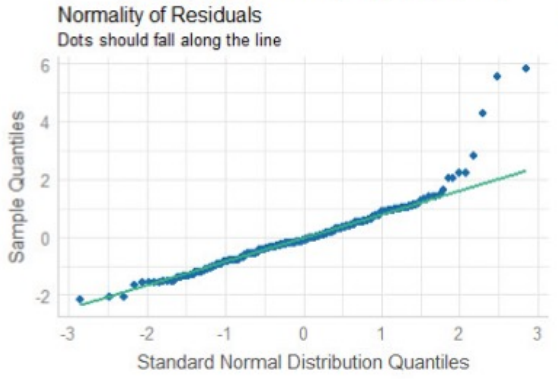


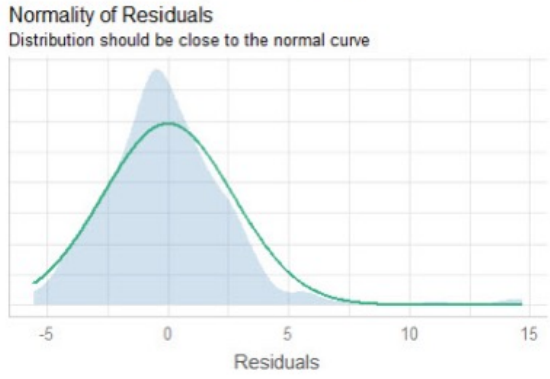












R-squared

R2

Motivation: R^2 measures how well a model performs compared to the constant mean baseline. Ideally, R^2 should be 1, ie the model would explain all the variance. However, the same results can be obtained with regression created with just 2 points. Thus we need p-value to inform how significant is that result

$$R^2 = 1 - \frac{RSS(model)}{RSS(mean)} \qquad R^2 = \frac{RSS(mean) - RSS(model)}{RSS(mean)}$$

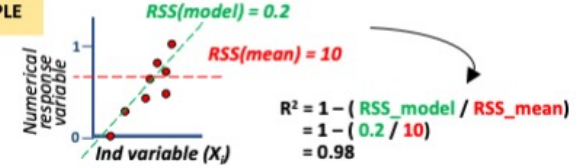
ABOUT

- **Always R^2 ≤1**
- **Ideally, R^2 should be close to one**
- **The coefficient is defined as the proportion of variance explained by the model eg: 85%**

VALUES

- **R^2 = 1,** *ie exactly 1 , then both the model and the baseline make make no mistakes or it is close to 0.*
- **R^2 → 1**; *the model preforms much better then the baseline mean*
- **R^2 ~0**; *the model performs similar to baseline mean*
- **R^2 <0**; *the model performs worst then the mean baseline*

EXAMPLE



R² = 1 – (RSS_model / RSS_mean)
= 1 – (0.2 / 10)
= 0.98

MEANING and Q&A

$$R^2 = 1 - \frac{\text{reduction in variance when we take the data into account (using the model)}}{\text{variation in y, taking into account only the results (y) i.e the model is simply a mean baseline}}$$

Are Low R-squared Values Always a Problem?

No! Regression models with low R-squared values can be perfectly good models for several reasons. Eg: some fields of study have an inherently greater amount of unexplainable variation. In these areas, your R2 values are bound to be lower. For example, studies that try to explain human behavior generally have R2 values less than 50%. Fortunately, if you have a low R-squared value but the independent variables are statistically significant, you can still draw important conclusions about the relationships between the variables. Statistically significant coefficients continue to represent the mean change in the dependent variable given a one-unit shift in the independent variable.

When it is a problem ?

There is a scenario where small R-squared values can cause problems. If you need to generate predictions that are relatively precise (narrow prediction intervals), a low R2 can be a showstopper

How high does R-squared need to be for the model to produce useful predictions?

That depends on the precision that you require and the amount of variation present in your data. A high R2 is necessary for precise predictions, but it is not sufficient by itself

Are High R-squared Values Always Great?

No! A regression model with a high R-squared value can have a multitude of problems, eg. there can be non-linear relationship between X and y. You can easily detect it with residual analysis.

R2 LIMITATIONS & PROBLEMS

Two Important Limitations

You cannot use R-squared to determine whether the coefficient estimates and predictions are biased, which is why you must assess the residual plots

R-squared does not indicate if a regression model provides an adequate fit to your data. A good model can have a low R^2 value. On the other hand, a biased model can have a high R^2 value!

Two Important Problems

R-squared Inflation

R-squared increases every time you add an independent variable to the model. The R-squared never decreases, not even when it's just a chance correlation between variables. A regression model that contains more independent variables than another model can look like it provides a better fit merely because it contains more variables

SOLUTIONS:

(a) provide notation such as R^2_q , where q, indicates number of coefficients /weights in the model, including the intercept, to avoid comparing models with R^2 with vastly different number of features.

(b) Residuals from two models can e compared with F-test

(c) Use Adjusted R^2

Overfitting

It may produces deceptively high R-squared values

ADJUSTED R2

The use of an adjusted R^2 (one common notation is $\bar{R}^2$, pronounced "R bar squared"; another is R^2_a or R^2_{adj}) is an attempt to account for the phenomenon of the R^2 automatically and spuriously increasing when extra explanatory variables are added to the model. There are many different ways of adjusting (see [13] for an overview). By far the most used one, to the point that it is typically just referred to as adjusted R, is the correction proposed by Mordecai Ezekiel.[13][14] The adjusted R^2 (according to Ezekiel) is defined as

$$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

where p is the total number of explanatory variables in the model (not including the constant term), and n is the sample size. It can also be written as:

$$\bar{R}^2 = 1 - \frac{SS_{\text{res}}/\text{df}_e}{SS_{\text{tot}}/\text{df}_t}$$

where df_t is the degrees of freedom $n-1$ of the estimate of the population variance of the dependent variable, and df_e is the degrees of freedom $n-p-1$ of the estimate of the underlying population error variance.

The adjusted R^2 can be negative, and its value will always be less than or equal to that of R^2. Unlike R^2, the adjusted R^2 increases only when the increase in R^2 (due to the inclusion of a new explanatory variable) is more than one would expect to see by chance. If a set of explanatory variables with a predetermined hierarchy of importance are introduced into a regression one at a time, with the adjusted R^2 computed each time, the level at which adjusted R^2 reaches a maximum, and decreases afterward, would be the regression with the ideal combination of having the best fit without excess/unnecessary terms.

P-value for R2

(Step 1) calculate F-value

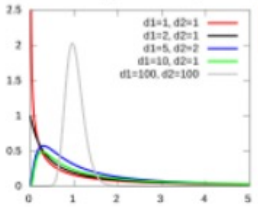
- F value, the critical value used to calculate p-value, with F-distribusion

$$F = \frac{RSS(mean) - RSS(model). / (p\_model - p\_means)}{RSS(model) / (n\text{-}p\_model)}$$

- ***where p_model/means*** *are degrees of freedom & it turns sum of squares into variance*
  eg: in the model y = 2x + 5
  p_model = 2
  in the mean, y = mean(y)
  p_mean = 1
- ***n: number of datapoints, Intuition behind:*** *then more dimensions in a feature space you use, then more data points you need to explain them.*
- *Example values:*
  - *Eg: F=2.41, F=4.56*

(Step 2) We use F-distribution with the corresponding degr. of freedom, for find p-value

- *F-distribution have two parameters d1, and d2*
  - *D1 = (P_model – p_mean)*
  - *D2 = (n-p_model)*
- *When d1 is large, the mean modes toward the "right", and it is large for models with many coefficients.*
- *d2, decides on the kurtosis, the larger it is, the higher the is the kurtosis (lighter tails). d2, increases with sample number.*



Why do we use F-distribusion

NOTATION

Vars	R-Sq	R-Sq(adj)
1	72.1	71.0
2	85.9	84.8
3	87.4	85.9
4	89.1	82.3
5	89.9	80.7

- Box –

Regression validation:

a process of testing whether the regression model are acceptable/good enough representation of the true underlying model that generated the data with some noise. Because we do not know, that true un. model we can use several tests instead:

- Analyse the goodness of fit of the regression; eg R^2 coeff. of determination, and test its statistical significance with F-test
- Perform residual analysis; eg. are they normally, and randomly
- Test regression model assumption see my slides on that)
- Use test data

TESXT ON Q&A and in R2 Limitations from https://statisticsbyjim.com
Some other elements from Wikipedia

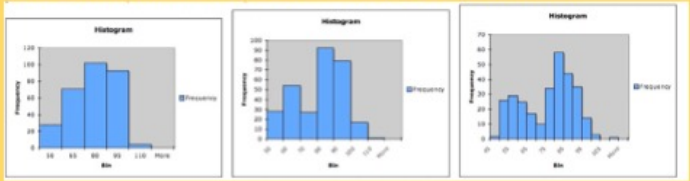
TESTING NORMALITY

VISUAL EXAMINATION

Histogram vs Probability Plots

A histogram (whether of outcome values or of residuals) is *not* a good way to check for normality, since histograms of the same data but using different bin sizes (class-widths) and/or different cut-points between the bins may look quite different (see example below.

>> Instead, use a *probability plot* (also know as a *quantile plot* or *Q-Q plot*). *Caution*: Probability plots for small data sets are often misleading; it is very hard to tell whether or not a small data set comes from a particular distribution.



INTEPRETATION

(a) Two Identical distributions:

➢ the Q–Q plot follows the 45° line y = x.

(b) Two distributions agree after linearly transforming the values in one of the distributions,

➢ Q–Q plot follows some line, but not necessarily the line y = x.

(c) the distribution plotted on the x-axis is more dispersed than the distribution plotted on the vertical axis.

➢ the general trend of the Q–Q plot is flatter than the line y = x,

(d) the distribution plotted on the y-axis is more dispersed than the distribution plotted on the horizontal axis.

➢ Q–Q plot is steeper than the line y = x

(e) one of the distributions is more skewed than the other, or that one of the distributions has heavier tails than the other.

➢ Q–Q plots are often arced, or "S" shaped

Although a Q–Q plot is based on quantiles, in a standard Q–Q plot it is not possible to determine which point in the Q–Q plot determines a given quantile. For example, it is not possible to determine the median of either of the two distributions being compared by inspecting the Q–Q plot. Some Q–Q plots indicate the deciles to make determinations such as this possible.

PROBABILITY PLOTS

DEF: graphical technique for assessing whether or not a data set follows a given distribution such as the normal or Weibull. The data are plotted against a theoretical distribution in such a way that the points should form approximately a straight line.

1. P-P PLOT

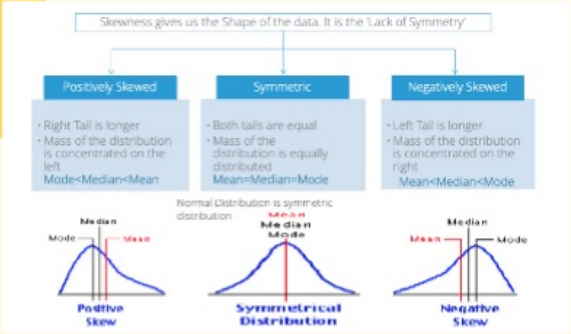
- "Probability-Probability" or "Percent-Percent" plot
- USED
  - to asses skenwness of the distribution
- LIMITATION: it is only useful for comparing probability distributions that have nearby or equal location. if two distributions are separated in space, the P–P plot will give very little data

2. Q-Q PLOT

- "Quantile-Quantile" plot
- **NORMAL PROBABILITY PLOT -** a Q–Q plot against the standard normal distribution

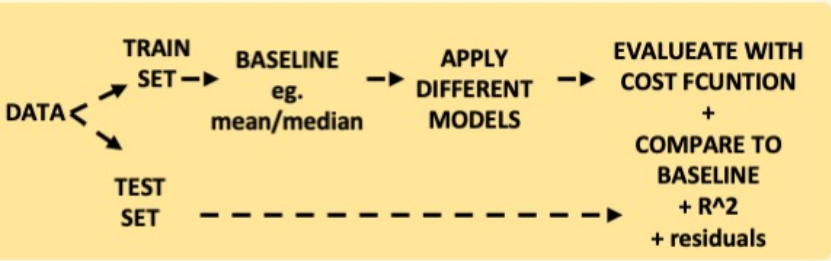
- DEF:
  - A Q–Q plot is a plot of the quantiles of two distributions against each other, or a plot based on estimates of the quantiles. The pattern of points in the plot is used to compare the two distributions.
  - Non-parametric approach to compare two datasets distribusiosn

- USED
  - plots the two cumulative distribution functions against each other.
  - compare two theoretical distributions to each other.
  - Or to compare collections of data, or theoretical distributions, eg. normal distribution
- MAIN ADVANTAGE
  - more widely used then PP-plots
  - Since Q–Q plots compare distributions, there is no need for the values to be observed as pairs, as in a scatter plot, or even for the numbers of values in the two groups being compared to be equal.
- RESULTS PROVIDED
  - Q-Q plot allows comparing the shapes of distributions, providing a graphical view of location, scale, skewness are similar or different in the two distributions.



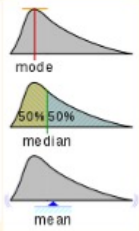
https://realpython.com/numpy-scipy-pandas-correlation-python/#rank-correlation
https://docs.tibco.com/data-science/GUID-75AB887A-9927-4772-BC83-DA09E65B3387.html
https://statisticsbyjim.com/regression/variance-inflation-factors/
https://medium.com/geekculture/holy-grail-for-understanding-all-the-assumptions-of-linear-regression-210f224192b5
https://stackoverflow.com/questions/42658379/variance-inflation-factor-in-python

Model Evaluation



Baseline

- **MODE/MOST FREQUENT;** categorical data, imbalanced dataset
- **MEDIAN;** for data with outliers, or skewed distribution, eg. gains
- **MEAN;** most used type of baseline, use with RSS error & data with no outliers



DummyRegressor

```
from sklearn.dummy import DummyRegressor

# Create the DummyRegressor object
dummy = DummyRegressor(strategy='mean')

# Fit the estimator
dummy.fit(x[:, np.newaxis], y);

# Vector with predictions from the baseline
pred_baseline = dummy.predict(x[:, np.newaxis])

rmse_baseline = RMSE(y, pred_baseline)
rmse_baseline # Returns: 371.11459394676217
```

Compute the mean/median and use it as the model to get the cost

```
# Compute baseline
pred_baseline = np.mean(y) # equals to 674.7218543
rmse_baseline = RMSE(y, pred_baseline)
rmse_baseline # 371.11
```

sklearn.dummy: Dummy estimators

User guide: See the Metrics and scoring: quantifying the quality of predictions section for further details.

dummy.DummyClassifier(* [, strategy, ...])	DummyClassifier is a classifier that makes predictions using simple rules.
dummy.DummyRegressor(* [, strategy, ...])	Regressor that makes predictions using simple rules.

Train/test split

```
x (50,)
y (50,)
train_data_ratio = 0.5

#### .... Divide X, Y into train-test datasets, .......

# * Generate a list of indexes
n          = len(x)
indexes    = np.arange(n)

# * shuffle
np.random.seed(0)
np.random.shuffle(indexes)

# * Split into train/test indexes
split_idx = int(n*train_data_ratio) # use int, to avoid having floats!
train_idx = indexes[:split_idx]
test_idx  = indexes[split_idx:]

# * Split data
x_tr, y_tr = x[train_idx], y[train_idx]
x_te, y_te = x[test_idx], y[test_idx]
```

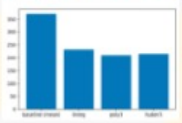
- **np.random.seed(0)**
- **np.random.shuffle()**

train_test_split()

```
from sklearn.model_selection import train_test_split

# * VERSION 1. ..... USING TRAIN-TEST RATIO
x_tr, x_te, y_tr, y_te = train_test_split(
                                          x, y,
                                          train_size  =0.7,
                                          test_size   =0.3,
                                          random_state=0)

                # IMPORTNAT - shullfling is done automatically,
```

Plot results with plt.bar()

```
# Bar chart
plt.bar([1, 2, 3, 4], [rmse_baseline, rmse_lr, rmse_poly3, rmse_huber3])
plt.xticks([1, 2, 3, 4], ['baseline (mean)', 'linreg', 'poly3', 'huber3'])
plt.show()
```



np.full_like()

Create a vector with the same value **np.full_like()**

```
# Vector with predictions from the baseline
pred_baseline = np.full_like(y, fill_value=np.mean(y), dtype=np.float)
pred_baseline # Returns: array([ 674.7218543, 674.7218543,  ...
```

%%time

```
%%time
np.random.rand(100,1).mean()

CPU times: user 82 µs, sys: 14 µs, total: 96 µs
Wall time: 90.1 µs
```

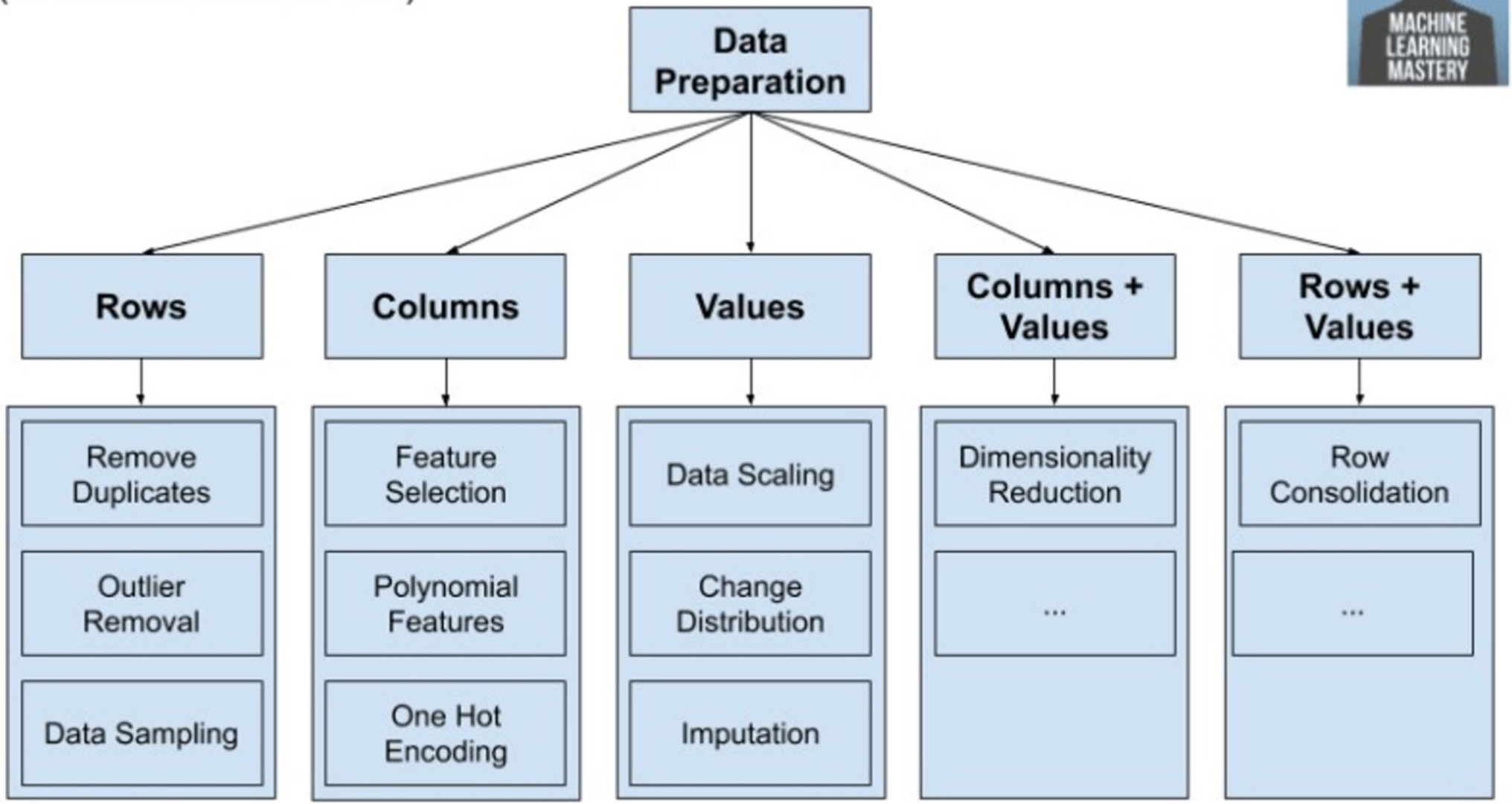
Used to check operation runtime

DATA PREPARATION

with

Sklearn

Data Preparation Framework
(*for structured/tabular data*)



Data Preprocessing

Feature Space & Dimensionality reduction

SimpleAI.ch

FEATURE ENG. - OVERVIEW

Feature engineering techniques

Numerical Range
- Scaling
- Normalizing
- Standardizing

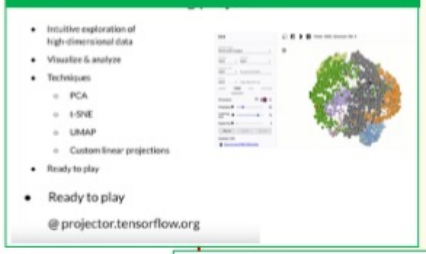
Grouping
- Bucketizing
- Bag of words

Other techniques

Dimensionality reduction in embeddings
- Principal component analysis (PCA)
- t-Distributed stochastic neighbor embedding (t-SNE)
- Uniform manifold approximation and projection (UMAP)

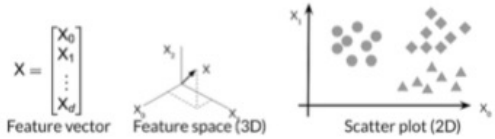
Feature crosses
- Combines multiple features together into a new feature
- Encodes nonlinearity in the feature space, or encodes

Online tools



FEATURE SPACE

- N dimensional space defined by your N features
- Not including the target label



Feature space

3D Feature Space

No. of Rooms X_0	Area X_1	Locality X_2	Price Y
5	1200 sq. ft	New York	$40,000
6	1800 sq. ft	Texas	$30,000

$Y = f(X_0, X_1, X_2)$

f is your ML model acting on feature space X_0, X_1, X_2

DIMENSIONALITY REDUCTION

- EDA
  - to test whether there is no batch effect
  - to find hidden clusters,
- Feature – extraction
  - Dimensionality reduction
  - To speed up the computation
  - simplifies the model
- Supervised ML:
  - to reduce overfitting/variance
  - increase the accuracy, of the model
- For feature selection

PCA

The idea behind PCA:
- PCA reduces the nr. of dimensions by projecting the data onto a set of n-ortogonal axes (90degr. to each other), called the PRINCIPIAL COMPONENTS. If we choose the set of "good axes" we can reduce the loss of information
- How: we create ne axes om a plot and project the overall variability in data from all dimensions in relation to that axis. A set of orthogonal axes with min. variance is selected.
- Good for analysis of high-dimensional data
- CAUTION: PCA alg. Favours features with the highest variance
- IMPORTANT: max incompetents == nr. Of features in the dataset.

Feature Granularity

Transformations	
Instance-level	**Full-pass**
Clipping	Minimax
Multiplying	Standard scaling
Expanding features	Bucketizing
etc.	etc.

Ensure feature space coverage !

- Train/Eval datasets representative of the serving dataset
  - Same numerical ranges
  - Same classes
  - Similar characteristics for image data
  - Similar vocabulary, syntax, and semantics for NLP data
- Data affected by: seasonality, trend, drift.
- Serving data: new values in features and labels.
- Continuous monitoring, key for success!

UMAP

- UMAP is a non-linear method for dimensionality reduction
- IDEA:
  - The dimensionality of many datasets is only artificial y high, and the data in these datasets depends on smalled number of confounding factors
  - UMAP alg. Is based on nearest neightbours alg.

When to transform the data

1. we can prepare entire dataset and then build the model

Pre-processing training dataset

Pros	Cons
Run-once	Transformations reproduced at serving
Compute on entire dataset	Slower iterations

2. we can do tranformation withitnn the model

Transforming within the model

Pros	Cons
Easy iterations	Expensive transforms
Transformation guarantees	Long model latency
	Transformations per batch: skew

BOX: Normalization vs Standarization

Normalization vs. standardization
- Normalization is good to use when:
  - the distribution of your data does not follow a Gaussian distribution.
  - This can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors and Neural Networks.
- Standardization,can be helpful:
  - the data follows a Gaussian distribution.
  - However, this does not have to be necessarily true.
  - Unlike normalization, standardization does not have a bounding range. So, even if you have **outliers** in your data, they will not be affected by standardization.
- You can always start by fitting your model to raw, normalized and standardized data and compare the performance for best results.
- Scaling of target values is generally not required.
- based on: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

T-SNE

UUSED FOR:
- EDA
- Visualization of high-dimensional data

The idea behin t-SNE
Converts similarities between data points into joint-probabilities Tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high dimensional data - In many cases t-SNE is the best method to visualize the dataset

Problems with t-SNE
- t-SNE had non-convex cost function == with different initialization points we will get different results. Therefore, if is best to use other dimensionality reduction method, eg
  - PCA for dense data
  - TruncatedSVD for sparse data
- Time consuming
  - you may reduce the nr. Dimensions with PCA and then use t-SNE
- **LIMITED TO FIT_TRANSFOMR**
  - t-.SNE an only be used to visualized or prepare the test data, but not to transform test data
  - Can not be used as data pre-processing step !

Data Preprocessing

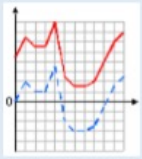
SKLEARN TRANFORMERS – PART 1

SimpleAI.ch

- FOR SCALING & CENTERING DATA FEATURES -

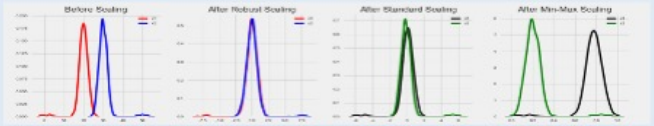
LINEAR TRANFORMATIONS

Each feature/column is transomed separately



Linear tranfs. **preserve** relative distance between features, and the shape of the distribution

IN PRACTICE: we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.



Original data
2 features with different means, and probably different SD

Robust Scaling
Both features have:
- MEAN = 0
- SD = IQR

Standard Sc.
Both features have:
- MEAN = 0
- SD = 1

MinMax Scaler
Features have DIFFERENT mean and SD, but
- Min value = 0
- Max value = 1

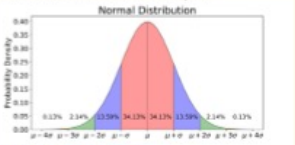
Centred around the same value , and expressed with the same unit & SD

Preserve data sparsity, and have the same range

SCALING

(i) **Used to Scale features with Gaussian distr.**
(ii) **Applied before using Gradient Descent,**
(iii) Applied before techniques that assume that assume normal distrib. Such as, linear regr., logistic regr. and linear discriminate analysis

z = (x-mean)/sd
sd = sqrt(Σ (x-mean)**2 / N)
- *Range [-∞, +∞]*
- *mean = 0*
- *sd = 1*




```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

 # you may use with_mean=False for sparse data

ROBUST SCALER

(i) Gives similar results as the standard scaler, but it based on median & IQR region size, instead of mean and SD.
(ii) Applied for data with **OUTLIERS**.
(iii) Can be used with different IQR size eg 5%, 25%, etc.. That may affect the results, and help scaling data to outliers.

X'i = (x-median)/IQR
- *IQR – difference between 25th and 75 percentiles (can be set as different value)*
- *Range [-∞, +∞]*
- *mean = 0*

```
>>> from sklearn.preprocessing import RobustScaler
>>> scaler = RobustScaler ().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
```

 # set different quintile range

```
>>> scaler = RobustScaler (quantile_range=(15, 85))
>>> scaler = RobustScaler (quantile_range=(value, 100-value)) # eg. value= 30
```

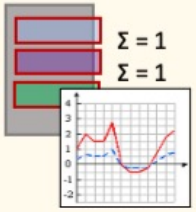
- TO SCALE SPARSE DATA -

NORMALIZATION

Scaling individual samples/rows to unit norm.
-> Sum of values in each observation (row) will be equal to 1 (unit norm).

(ii) Used for **SPARSE DATASETS** with attributes of varying scales
(iii) often used in **text classification and clustering contexts.**
(iv) alg. that use weights, eg NN, and alg, that use distance measures eg. kNN

x/(L1, L2 or Max norm)
- ***Range [0,1]***
- *Sum for each row (norm) = 1*
- ***Smoothing effect***



Normalizer

```
# {'l1', 'l2', 'max'}, default='l2
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer ().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
```

SCALING FEATURES TO A RANGE

Scaling features to lie between a given minimum and maximum value
eg: to convert a temperature from Celsius to Fahrenheit.

- **SPARSE DATA, especially MinMaxScaller** was designed for it.
  - Provide robustness to very small standard deviations of features
  - Do not centre values, and preserve zero entries in sparse data.
- **OUTLIERS**, but in that case there can be a problem (see caution)

Min-Max Scaler

Rescales features between 0-1

(i) Applied before using optimization alg. like **gradient descent**, that assume all values have 0-1 range
(ii) Used with alg. that use weight inputs; regression, NN.
(iii) and with alg. that use distance measures, k-NN

x' = (x – min) / (max – min)
- ***Range [0,1]***
- *If x = min(X), then x'=0*
- *If x = max(X), then x'=1*

CAUTION: may rescale values to very small intervals, if **outliers** are present

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> transformer = MinMaxScaler()
>>> rescaledX = transformer.fit_transform(X)
```

 # you may provide explicirt min/max values

```
>>> transformer = MinMaxScaler(feature_range=(0, 123))
```

MaxAbs Scaler

Rescales features between -1 & 1
Divides each value by maximum absolute value of each feature.

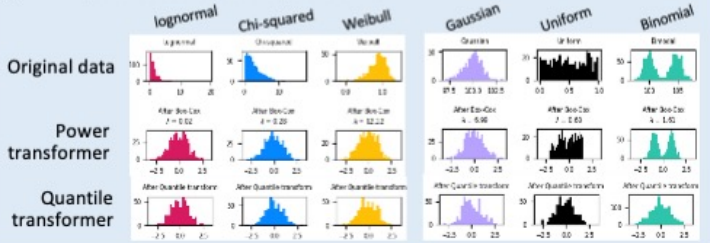
Range [-1,..,1]
NaN - remain NaN

```
>>> from sklearn.preprocessing import MaxAbsScaler
    X = [[ 1., -1., 2.], ... [ 2., 0., 0.], ... [ 0., 1., -1.]]
>>> transformer = MaxAbsScaler().fit(X)
>>> transformer.transform(X)
    array([[ 0.5, -1. , 1. ], [ 1. , 0. , 0. ], [ 0. , 1. , -0.5]])
```

- USED TO CORRECT SKEWNESS IN THE DISTRIBUSIONS -

NON-LINEAR TRANSFORMATIONS

(i) ***transforms the features to follow a uniform or a normal distribution***
(ii) To remove/reduced the impact of **OUTLIERS**
(iii) spread out the most frequent values



CAUTION:
(1). these methods don't scale the data to a predetermined range
(2). These methods may distort lin. corr. between var's measured at the same scale, but renders var's measured at different scales more directly comparable

Quantile transformer

Non-parametric approach

- Range: [0,1]
- spread out the most frequent values
- *transformed data is the approximation of the quantile position of the actual data*

How it works?
- first, cdf is used to map the original values to a uniform distribution.
- then, these values are mapped to output distribution with quantile function.
- values below or above the fitted range will be mapped to the bounds of the output distr.

```
>>> from sklearn.preprocessing import QuantileTransformer
>>> qt = QuantileTransformer(n_quantiles=10, random_state=0)
>>> qt.fit_transform(X)
```

	mpg	mpg_trans
0	18.0	0.286873
1	15.0	0.153652
2	18.0	0.286873
3	16.0	0.202771
4	17.0	0.239295

- **n_quantiles;** Typically large, default 1000.
- **output_distribution** *{'uniform', 'normal'},* distrib. used for the transformed data.
- **ignore_implicit_zeros** If True, zeros are not used, and stay as zeros.
- **Subsample** *int,* max nr of used samples

Power transformer

Parametric approach

- map data from any distribution to a Gaussian distribution
- stabilize variance
- minimize skewness.

Two methods:
'box-cox' - needs the data to be positive
'Yeo-Johnson' - data to be both negative and positive.

```
>>> pt = PowerTransformer(method='box-cox', standardize=False)
>>> X_lognormal = np.random.RandomState(616).lognormal(size=(3, 3))
>>> pt.fit_transform(X_lognormal)
```

Skilearn API

`fit`(X[, y])	Compute the median and quantiles to be used for scaling.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(X)	Scale back the data to the original representation
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Center and scale the data.

Data Preprocessing

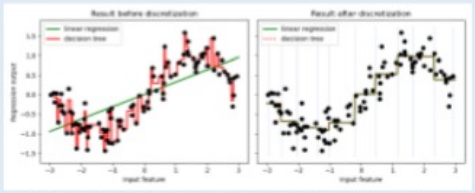
SKLEARN TRANFORMERS – PART 2

SimpleAI.ch

DISCRETISATION/BINNING

(i) Used to discretize continuous features
(ii) May improve linear models,
(iii) Reduce variance in non-linear models, like regression trees,

After discretization, linear regression and decision tree make exactly the same prediction



https://scikit-learn.org/stable/auto_examples/preprocessing/plot_discretization.html#sphx-glr-auto-examples-preprocessing-plot-discretization-py

binarization

All values > threshold are 1 and all ≤threshold are marked as 0

(i) **Applied for PROBABILITIES**
(ii) Used for Feature eng.

Generates 0 or 1 values only

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X_train)
```

K-bins discretization

Transforms continuous feature into a categorical feature by partitioning it into several bins within the expected value range (intervals).

- ***0 -K-1 bins***
- ***Ecoding***
  - ***One-hot-enc***
  - ***ordinal (0, 1, 2, …, k-1)***

Value	Bins
10	1
15	1
20	1
25	2
30	2

 # Encoding

- ***'onehot',: default, i***gnored features, eg missing data, novelties, are always stacked to the right
- ***'ordinal':*** return sthe bin identifier encoded as an integer value

CAUTION
Ordinal values may still be used, in most models

 # Strategy used to define the widths of the bins.

- **Uniform;** All bins in each feature have identical widths (max-min values).
- **Quantile;** All bins in each feature have the same number of points.
- **kmeansValues;** in each bin have the same nearest center of a 1D k-means cluster.

```
>>> from sklearn.preprocessing import KBinsDiscretizer
>>> enc = KBinsDiscretizer(n_bins=10, encode='onehot')
>>> X_binned = enc.fit_transform(X)
```

ENCODING CATEGORICAL FEATURES & TARGETS

one-hot/dummy encoding

Nan & None
- Considered as separate values,

handle_unknown='ignore'
- If ignore, these are mapped to zeros ,

drop='first'
- column with the 1st category is dropped,
- If only one cat is present, it is also dropped,
- Dropped column contains zeros, representing unknown variables (novelties)

Why to drop a column?
Done to avoid co-linearity in the input matrix in some classifiers. Eg., non-regularized regression (LinearRegression), since co-linearity would cause the covariance matrix to be non-invertible

```
>>> from sklearn.preprocessing import OneHotEncoder()
>>> enc = OneHotEncoder(
            drop='first',
            handle_unknown='ignore' )
            # unknown/novelties = will be encoded with zero
>>> enc_X = enc.fit_transform(X)
>>> enc.categories_ # returns categories,

# you may specify explicit categories with 'categories' parameter
- NOT ADVISED
 >>> X =    [['male', 'uses Safari'], ['female', 'uses Firefox']]
 >>> genders  = ['female', 'male']
 >>> browsers  = ['uses Firefox', 'uses Safari']
 >>> enc = OneHotEncoder(categories=[genders, browsers])
```

OrdinalEncoder()

transforms each categorical feature to one new feature of integers (0 to n_categories - 1)

IMPORTANT
Ordinal values can not be used directly with all scikit-learn estimators, as these expect continuous input

 # handle_unknown & unknown_value
- If =='error', an error will be raised in case of novelty is detected.
- If =='use_encoded_value', alg will use value provided with unknown_value, None, will be returned in inverse_transform

 # NaN
- Stays NaN

```
>>> from sklearn.preprocessing import OrdinalEncoder()
>>> enc = OrdinalEncoder()
>>> encoded_X = enc.fit_transform(X_train)
```

get_dummies()

One-hot encoder from pandas

https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

Time related feature eng

https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html#sphx-glr-auto-examples-applications-plot-cyclical-feature-engineering-py

ADDING POLYNOMIAL FEATURES

PolynomialFeatures()

- **interaction_only**, if True, only the highest degree is used
- **include_bias** *If True, adds bias term, w0, with 1 only in each row.*
  For intercept

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(degree=3, include_bias=True)
>>> poly.fit_transform(X)
>>> poly_get_feature_names() # will retunr names of all new features
```

with FunctionTransformer() And labda function

```
>>> poly_columns = ['col_1', 'col_2']
>>> poly_transformer = Pipeline([
        ('scaler', StandardScaler()),
        ('poly', FunctionTransformer(
            lambda X: np.c_[X, X**2, X**3])  ])
```

- **It wont add bias term !**

CUSTOM TRANFORMERS

FunctionTransformer()

Used to implement a transformer from an arbitrary function with transformer API

 # Example 1. build a transformer with a log transformation

```
>>> import numpy as np
>>> from sklearn.preprocessing import FunctionTransformer
>>> transformer = FunctionTransformer(np.log1p, validate=True)
>>> transformer.fit_transform(X)
    # validate=True; check if the imput is an array

# ensure that inverse_tranform is possible
>>> enc = transformer.fit(X, check_inverse=True)
    # if not, returns a warning
    # Use fit, before transform, to apply it.
```

 # Example 2. provide stats for twitter posts, eg. post length, and sentence nr.

```
>>> def text_stats(posts):
...         return [{'length': len(text),
...                  'num_sentences': text.count('.')}
...                  for text in posts]
>>> text_stats_transformer = FunctionTransformer(text_stats)
```

 # Example 3. just add 1 to each value in a transformed columns

```
>>> def cust_func(x):
....        return x +1
```

LABEL ENCODERS

LabelEncoder()

 # Encodes labels/targets in y,

```
>>> from sklearn.preprocessing import LabelEncoder
>>>  enc = LabelEncoder()
>>>  y = enc.fit_transform(y)
```

LabelBinarizer()

- ***Works, like one-hot encoder,***
- ***Can be used with mutivariate regression***

Sourses

https://towardsdatascience.com/5-data-transformers-to-know-from-scikit-learn-612bc48b8c89
https://scikit-learn.org/stable/modules/preprocessing.html#spline-transformer
https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py

by **Pawel Rosikiewicz**

Outliers & Novelty Detection - basic

SimpleAI.ch

OUTLIERS & NOVELTIES

OUTLIERS
- an observation or event that deviates so much from other events to arouse suspicion it was generated by a different means, or distribution
- Measurement that is rare, distinct, or does not fit in some way.

NOVELTIES
- outliers, or unseen values /classes/categories in a new dataset,

CAUSES
- Measurement or input error.
- Data corruption.
- True outlier observation (e.g. Michael Jordan in basketball).
- Getting new, more diverse data,
- To small training dataset used

OUTLIER VS NOVELTY DETECTION

Techniques:
- Outlier det. == unsupervised anomaly det.
- Novelty det == semi-supervised anomaly det.

comments
- available estimators assume that the outliers are located in low density regions, ie: these are rare observations, with feature values different the other samples, and they can not form form a dense cluster.
- Novelties can form a dense cluster if they are in a low-density region of the training data

PROBLEM WITH IDENTYFYING OUTLIERS

There is no precise way to define and identify outliers in general because of the specifics of each dataset. Instead, you, or a domain expert, must interpret the raw observations and decide whether a value is an outlier or not. Even with a thorough understanding of the data, outliers can be hard to define. Great care should be taken not to hastily remove or change values, especially if the sample size is small.

WHAT TO DO NEXT

First, you should verify that the observation isn't a result of a data entry error or some other odd occurrence. If it turns out to be a legit value, you can then decide if it's appropriate to delete it, leave it be, or simply replace it with an alternative value like the median.

GOOD PRACTICE

Check if outliers can be identified systematically, eg: plot identified outlier values, against other predictors, and check if you can use some of them to identify your outliers and see how do they compare with

STANDARD DEVIATION

Assumptions

Gaussian or Gaussian-like distribution of values in the sample test for normal distribution (histogram, qq-plot, boxplot + stats)

Properties

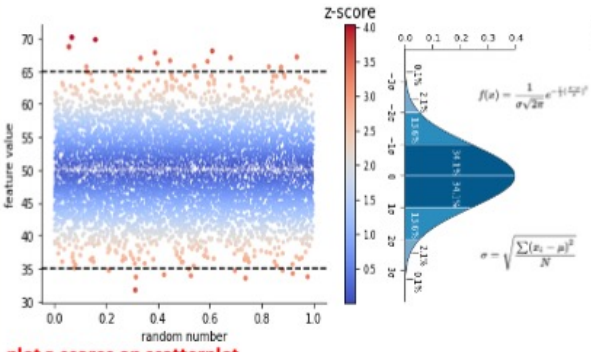
The Gaussian distribution has the property that the standard deviation from the mean can be used to reliably summarize the percentage of values in the sample
- 1 SD: 68% of values in the sample
- 2 SD : 95% ...
- 3 SD : 99.7% ...

So, if the mean is 50 and the standard deviation is 5, as in the test dataset above, then all data in the sample between 45 and 55 will account for about 68% of the data sample.

Outliers' definition
- **3SD**; typically threshold, A value that falls outside of 3 standard deviations. It is part of the distribution, but it is an unlikely or rare event at approximately 1 in 370 samples
- **2SD** can be used for smaller samples of data (95%)
- **4SD** can be used for large samples of data (99.9%)

Limitations

Method can be problem ting with data having a lot of zeros, such as basement area, or nothing, but not a missing data in that case.



plot z-scores on scatterplot

```
z_scores  = np.abs((data - np.mean(data))/np.std(data))
# scatter with z-value as the point colors,
plt.scatter(
   x=np.random.random(data.shape[0]), # random floats, [0,1]
   y=data,
   s=z_scores*3, # marker size - it will make outliers more visible
   c=z_scores, # nubers wil be used in combinaiton with selected cmap
   cmap=plt.cm.coolwarm # if not provided, automatic selection is given
)
plt.ylabel("feature value")
plt.xlabel("random number")
# colorbar
clb=plt.colorbar()
clb.ax.set_title('z-score',fontsize=14)
# add lines with ±3xSD/thresholds
plt.axhline(y=lower, color='black', linestyle='--')
plt.axhline(y=upper, color='black', linestyle='--')
sns.despine()
```

INTERQUANTILE RANGE

Method applied for samples with a non-Gaussian distribution

Properties
- **IRQ** - the difference between 25th and 75th percentile, is used to define the outliers. eg. if boxplot, have 25th%=10, and 75%=16, IQR =6. IQR is shown as box on boxplot,
- **Threshold** is calculated by adding or subtracting k*IQG, from the 25th of 75th quantile,

Outliers' definition
- 1.5 IQR – commonly used
- 3 IQR – "far outs", ie extreme outliers,

Example
- Lets have the following: 25th%=10, and 75th%=16,
- Then, IQR =6 = box on a boxplot, with line plotted on 50th%
- Upper threshold = (10-6*1.5) = 1 - upper fence of the boxplot whisker
- Lower threshold = (16+6*1.5) = 25 – lower fence of the boxplot wiser,

How to find percentiles,
- Fist, sort all values,
- Find 50th percentile, is the number of values is even, use the mean of the wo values,
- Use the same approach for other percentiles,
- Numpy: **>>> percentile()**

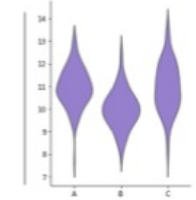
HOW TO TREAT OUTLIERS

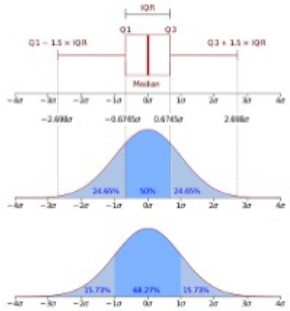
First, you should verify that the observation isn't a result of a data entry error or some other odd occurrence.
If it turns out to be a legit value, you can then decide if it's appropriate to delete it, leave it be, or simply replace it with an alternative value like the median.

EFFECTS
- On cost function results, Reg: RSS error function puts too much weight on residuals calculated with points far from predictions

SOLUTIONS
- Remove them (eg see below, z-score)
- Replace outliers with mean/median or trim their values to some min/max values
- Use cost function like Huber Loss
- Introduce weight into the model:
  eg: z-score + threshold

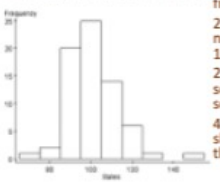
- Box - Violin plots





- Box - Histograms

Histograms are good for showing general distributional features *of dataset variables. You can see roughly where the peaks of the distribution are, whether the distribution is skewed or symmetric, and if there are any outliers.*
- **is the distr. Unimodal** (one peak), **bimodal** (two peaks), or **uniform** (caution it may depend on the # of bins)
- ... **symmetric or asymmetric**
- ... **skewed,** if yes, is it skewed left or right
- are they **outliers**
- what is the **span**, or the difference between the min and max values

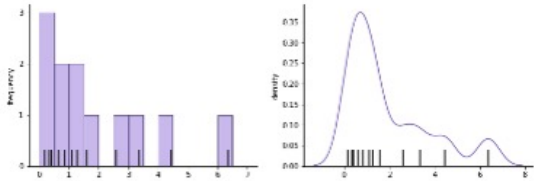


Data Interpretation example

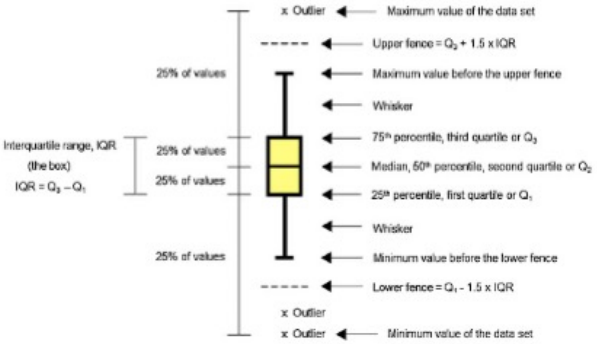
1. This newspaper typically sold about 100,000 copies per day. Sales between 90,000 and 110,000 were quite frequent.
2. For this sample of 70 days' sales, the smallest number of newspapers sold was about 70,000 and the largest is about 150,000.
2. There were an unusually large number of newspapers sold one day. The day on which 150,000 newspapers were sold is atypical.
4. Finally, due to the atypical large value, the histogram is slightly skewed to the right, or positively skewed. Without this value, the histogram would be reasonably symmetric.

- Box - Density curve / Kde-plots

- A density curve, or kernel density estimate (KDE),
- an alternative to the histogram that gives each data point a continuous contribution to the distribution.
- In a histogram, you might think of each data point as pouring liquid from its value into a series of cylinders below (the bins). In a KDE, each data point adds a small lump of volume around its true value, which is stacked up across data points to generate the final curve. The shape of the lump of volume is the 'kernel', and there are limitless choices available. Because of the vast amount of options when choosing a kernel and its parameters, density curves are typically the domain of programmatic visualization tools.



- Box - Boxplot



Some materials taken from Wikipedia,
Nice blogpost on histograms; https://chartio.com/learn/charts/histogram-complete-guide/
Sklearn, excellent introduction to outlier and novelty detection; https://scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection
How to Remove Outliers for Machine Learning; https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/
4 Automatic Outlier Detection Algorithms in Python; https://machinelearningmastery.com/model-based-outlier-detection-and-removal-in-python/

by Pawel Rosikiewicz

Outliers & Novelty Detection – high-dimensional data

SimpleAI.ch

Cook's Distance

How to check this assumption: The most common way to test for extreme outliers and influential observations in a dataset is to calculate Cook's distance for each observation. If there are indeed outliers, you can choose to (1) remove them, (2) replace them with a value like the mean or median, or (3) simply keep them in the model but make a note about this when reporting the regression results.

Cook's distance, often denoted D_i, is used in regression analysis to identify influential data points that may negatively affect your regression model.

The formula for Cook's distance is:

$$D_i = (r_i^2 / p{*}MSE) * (h_{ii} / (1-h_{ii})^2)$$

where:

- r_i is the i^{th} residual
- **p** is the number of coefficients in the regression model
- **MSE** is the mean squared error
- h_{ii} is the i^{th} leverage value

Although the formula looks a bit complicated, the good news is that most statistical softwares can easily compute this for you.

Essentially, Cook's Distance does one thing: **it measures how much all of the fitted values in the model change when the i^{th} data point is deleted.**

A data point that has a large value for Cook's Distance indicates that it strongly influences the fitted values. A general rule of thumb is that any point with a Cook's Distance over 4/n (*where n is the total number of data points*) is considered to be an outlier.

It's important to note that Cook's Distance is often used as a way to *identify* influential data points. Just because a data point is influential doesn't mean it should necessarily be deleted – first you should check to see if the data point has simply been incorrectly recorded or if there is something strange about the data point that may point to an interesting finding.

In Python https://www.statology.org/cooks-distance-python/

Example of using cooks distance to logistic regression (in R) http://www.sthda.com/english/articles/36-classification-methods-essentials/148-logistic-regression-assumptions-and-diagnostics-in-r/

Copied from Wikipedia

Automated algorithms for outlier detection

Robust Covariance

MDC – Min. Cov. Determinant

***EllipticEnvelope**(contamination=.1)*

assumes the data is Gaussian and learns an ellipse

Pros

- Robust, and fast method for outlier detection
- If applied to normally distributed random var. a simple stat methods can be used to identify potential outliers/novelties

Cons & Limitations

- degrades when the data are not unimodal,
- … or not-normally distributed.
- MCD estimator has low error rate provided that n_samples > 5 n _features

Comment

- Robust == method estimates covariance without strong influence of outliers, using MCD-based Mahalanobis distances that provides better results then MLE

Isolation forest (iForest)

***IsolationForest**(contamination=0.1)*

based on ensemble ExtraTreeRegressor method

iForest isolates observations by randomly selecting a feature and then randomly selecting a split value between min/max of the selected feature. The nr of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is a measure of normality and our decision function. Random partitioning produces noticeably shorter paths for anomalies.

Pros

- Faster to train with big data
- Good for high-dimensional data
- Can handle multi-modal data sets

Cons

- Not easily interpretable;
- High memory with large dataset

OneClass SVM

***OneClassSVM**(nu=0.1)*

SGDOneClassSVM ()

Traditional SVM

- Project point to a higher dimensional space
- separates classes using a hyperplane with the largest possible margin.
- to avoid over-fitting, it allows some soft margin for some points to lie in the region between support vectors
- objective: maximise the margin

One class SVM

- Project point to a higher dimensional space
- Separate all the data points from the origin in the feature space using hyper-plane/hyper-sphere
- use soft margin for smoothness, - we use a parameter (nu) to allow fixed fraction of outliers
- objective: maximise the distance between the hyper-plane and the origin or minimise the volume of the hyper-sphere (two authors described it differently)

Pros

- Best use for novelty detection, with high-dimensional data, and with train set not contaminated with outliers.
- SDG, implementation works well with large datasets, (not shown below, and didn't work on my older python version)

Cons

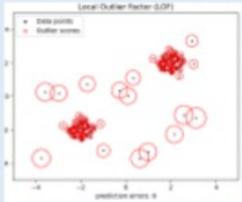
- generally, not good for outlier detection - SVM by itself is sensitive to outliers
- Hyperparameter tuning may be challenging

LOF – local Outlier Factor

***LocalOutlierFactor**(novelties=True)*

Based on knn alg

(LOF) alg. computes a score (called local outlier factor) reflecting the degree of abnormality of the observations. It measures the local density deviation of a given data point with respect to its neighbours. The idea is to detect the samples that have a substantially lower density than their neighbours. LOF score of an observation == the ratio of the avg. local density of his k-nearest neighbors, and its own local density: a normal instance is expected to have a local density similar to that of its neighbors, while abnormal data are expected to have much smaller local density.



Pros

- Fast, and low memory,
- takes both local and global properties of datasets into consideration: ie LOF perform well even in datasets where abnormal samples have different underlying densities. The question is not, how isolated the sample is, but how isolated it is with respect to the surrounding neighborhood.

Cons:

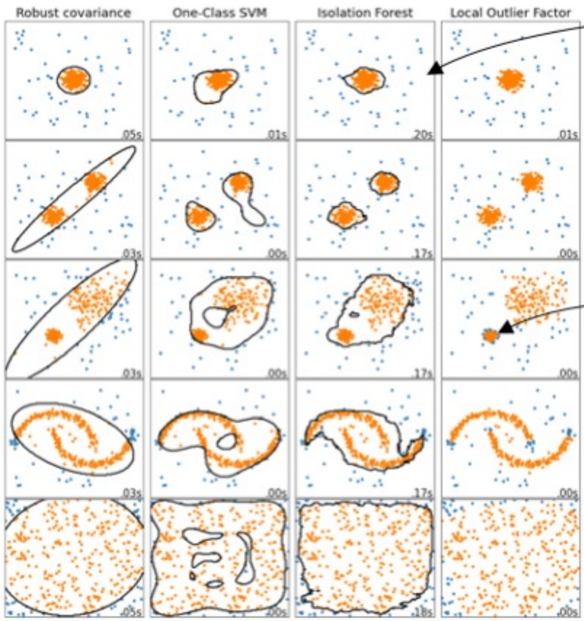
- Generally, for lower number of dimensions, then iForest

Comments

- **n_neighbors=20** work well in general.
- When the proportion of outliers is high (i.e. >10), n_neighbors should be greater
- **Novelty=True**, onlky then you can use LOF for finding novelties

How it was made

For each dataset, 15% of samples are generated as random uniform noise. This proportion is the value given to the nu parameter of the OneClassSVM and the contamination parameter of the other outlier detection algorithms. Link to code in sources, and on my github with credits to authors.



Decision boundary

*between inliers and outliers are displayed in black except for Local Outlier Factor (LOF) as it has no predict method to be applied on new data when it is used for outlier detection, unless used with **novelties=True***

Node

Region of high density

- **fit(X):** fit/train the model with data set \`X\`.
- **predict(X):** determine whether the observations in \`X\` are inliers \`1\` or outliers \`-1\`.
- **decision_function(X):** score/metric used to determine whether a point is an inlier/outlier. The interpretation of the output of the \`decision_function\` method is algorithm specific

SOURCES

https://scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection
https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_anomaly_comparison.html#sphx-glr-auto-examples-miscellaneous-plot-anomaly-comparison-py
https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/
https://machinelearningmastery.com/model-based-outlier-detection-and-removal-in-python/
https://www.statology.org/assumptions-of-logistic-regression/
https://www.statology.org/how-to-identify-influential-data-points-using-cooks-distance/
https://medium.com/mlearning-ai/anomaly-detection-with-machine-learning-8fa942fb5adc
Minimum covariance Distance and estimator; https://scikit-learn.org/stable/auto_examples/covariance/plot_robust_vs_empirical_covariance.html#sphx-glr-auto-examples-covariance-plot-robust-vs-empirical-covariance-py
iForest xample; https://scikit-learn.org/stable/auto_examples/ensemble/plot_isolation_forest.html#sphx-glr-auto-examples-ensemble-plot-isolation-forest-py

Data Preprocessing

IMPUTATION WITH SKLEARN

SimpleAI.ch

MISSING VALUES

DELETION — *Pandas functions/pre-processing*

- **PAIRWISE DELETIONS** — Deleting only missing data in compared cases
- **LIKEWISE DELETIONS** — dropping entire rows, with missing values
- **DROPPING COLUMNS** — dropping entire columns, with missing values

ML alg's require numeric input values, and a value to be present for each row and column in a dataset. It is common to identify missing values and replace them with a numeric value.

IMPUTATION

- **ADVANCED**
  - **kNN Based** — "nearest neighbor imputation" **kNN model** is used to predict the missing values, although any other model can also be applied. *IterativeImputer()*
  - **MICE** — MICE; multiple imputation method used to replace missing data **values** in a data set under certain assumptions about the data missingness mechanism (e.g., the data are missing at random, the data are missing completely at random). *KNNImputer()*
- **GENERAL**
  - **Constant**
  - **Non missing = 1, Missing = 0** — *MissingIndicator()*
  - **Mean, Median, Mode, Most Frequent** — *SimpleImputer()*
- **TIME SERIES**
  - **Forward Fill**
  - **BackFill**
  - **Linear Interpolation** — *Pandas functions*

SimpleImputer()

Imputation transformer for completing missing values

- **missing_values;** defining values considered as missing in a dataset
- **Strategy;** *default='mean'*
  - **"mean"/ "median"**
  - **"most_frequent";** works for both, str, and numeric data, Caution, if >1 value, the smallest is used.
  - **"constant";** you must define **fill_value.**
- **add_indicator,** If True, a MissingIndicator transform will stack onto output of the imputer's transform. This allows a predictive estimator to account for missingness despite imputation. If a feature has no missing values at fit/train time, the feature won't appear on the missing indicator even if there are missing values at transform/test time.

```
>>> fomr sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(
            missing_values=0,
            strategy='mean',
            axis=0)
>>> imp.fit_transform(X_train)
```

KNNimputer()

(i) kNN imputation (K-nearest neighbor model)

- *(i) A new sample is imputed by finding the samples in the training set "closest" to it and averages these nearby points to fill in the value.*
- *(ii) More efficient then the commonly used row average method or replacing nan with Zeros*

(ii) Alternatively: **regression models** can be used for numeric variables,

(iii) Configuration of KNN imputation involves selecting:

- (iii) **distance measure** (e.g. Euclidean)
- **(iv) k hyperparameter,** ie. the number of contributing neighbors for each prediction

```
>>> from sklearn.impute import KNNImputer
>>> imputer = KNNImputer(
...            n_neighbors=5,
...            weights='uniform',
...            metric='nan_euclidean')
...                    # nan avare method, do not use nan to compute distances,
...                    # if other methods – you must provide distance matrices,
...                      or your function
>>> imputer.fit(X)
>>> Xtrans = imputer.transform(X)
```

GOOD PRACTICE

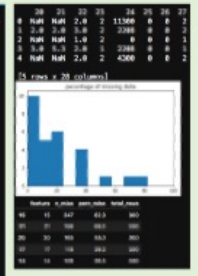
Convert missing values to np.nan in pandas.df that may have pd.NA

Example: Tuning KNNinputer

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from numpy import isnan
from sklearn.impute import KNNImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import KNNImputer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
```

1. Load data, EDA on missing data points

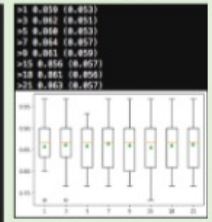
```
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/horse-colic.csv'
dataframe = pd.read_csv(url, header=None, na_values='?')

# summarize the first few rows
print(dataframe.head())

# summarize the number of rows with missing values for each column
results=[]
for i in range(dataframe.shape[1]):
    # count number of rows with missing values
    n_miss = int(dataframe[[i]].isnull().sum())
    perc = float(n_miss/dataframe.shape[0]*100)
    results.append({"feature":i, "n_miss":n_miss,
                    "perc_miss":np.round(perc,1),
                    "total_rows":dataframe.shape[0]})

# show all features on histogram
fig, ax = plt.subplots();
ax.hist(pd.DataFrame(results).perc_miss.values)
ax.set_xlim(0,100)
ax.set_title("pecentage of missing data")
plt.show();

# display columns with the highest pers/nr of missing values
pd.DataFrame(results).sort_values("perc_miss", ascending=False).head()
```



2. Find best k, suing k-fold cv, and random forest classifier

```
# split into input and output elements
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]

# define modeling pipeline
model = RandomForestClassifier()
imputer = KNNImputer()

# evaluate each strategy on the dataset
results = list()
strategies = [str(i) for i in [1,3,5,7,9,15,18,21]]  # k for KNNimputer
for s in strategies:
    # create the modeling pipeline
    pipeline = Pipeline(steps=[
        ('i', KNNImputer(n_neighbors=int(s))),
        ('m', RandomForestClassifier())
    ])

    # evaluate the model
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

    # store results
    results.append(scores)
    print('>%s %.3f (%.3f)' % (s, np.mean(scores), np.std(scores)))

# plot model performance for comparison
plt.boxplot(results, labels=strategies, showmeans=True)
plt.show()
```

```
>1 0.859 (0.053)
>3 0.862 (0.051)
>5 0.860 (0.053)
>7 0.864 (0.057)
>9 0.861 (0.059)
>15 0.856 (0.057)
>18 0.861 (0.056)
>21 0.863 (0.057)
```



3. Create pipeline with selected steps& hyperparameters, fit it, and use for prediction for new data

```
# define new data
row = [2, 1, 530101, 38.50, 66, 28, 3, 3,
       np.nan, 2, 5, 4, 4, np.nan, np.nan, np.nan,
       3, 5, 45.00, 8.40, np.nan, np.nan,
       2, 11300, 00000, 00000, 2]

# create the modeling pipeline
pipeline = Pipeline(steps=[
    ('i', KNNImputer(n_neighbors=21)),
    ('m', RandomForestClassifier())
])

# fit the model
pipeline.fit(X, y)

# make a prediction
yhat = pipeline.predict([row])
print('Predicted Class: %d' % yhat[0])
```

```
Predicted Class: 2
```

The plot suggest that there is not much difference in the k value when imputing the missing values, with minor fluctuations around the mean performance (green triangle).

Example taken from: https://machinelearningmastery.com

APPLYING CUSTOM TRANSFORMERS

EXAMPLE: APPLYING CUSTOM TRANFORMERS

```
>>> import numpy as np
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import FunctionTransformer
>>> from sklearn.preprocessing import OneHotEncoder, StandardScaler, KBinsDiscretizer
>>> from sklearn.compose import ColumnTransformer

# 1. DEFINE CUTOM TRANFROMERS
# create custom transformer
'' unlike pipeline, make_pipepline
  generates names for steps automatically
  ->> lowercase name of an estimator
  otherwise, these two fucntions work in the same way
'''
>>> log_scale_transformer = make_pipeline(
...                         FunctionTransformer(np.log, validate=False),
...                         StandardScaler()
...                         )

# 2. DEFINE PREPROCESSING FUNCTION WITH DIFFERENT  TRANFORMATIONS
      FOR DIFFERENT COLUMNS
# Create final preprocessor for the data, with ColumnTransformer()
''' ColumnTransformer takes a list with instrucitons for each column/col. Group.
   for each of them you must provide the folloiwing
            (i)  unique preprocessor name
            (ii). Transformer function, "passthrough" str, to not make any modifications
                        - if more then one function, you need to create it separately,
                          using pieline or make pipline,
                          like log_scale_transformer in this example.
            (iii) column names in input data, that will be transformed  (LIST)
'''
linear_model_preprocessor = ColumnTransformer(
   transformers=[

      ("passthrough_numeric", "passthrough", ["column_1"]),

      ("binned_numeric", KBinsDiscretizer(n_bins=10), [" column_2", " column_ 3"]),

      ("log_scaled_numeric", log_scale_transformer, [" column_4"]),

      ("onehot_categorical", OneHotEncoder(), [" column_5", " column_5", " column_6"]),
   ],
   remainder="drop",     # TWO OPTION {'drop', 'passthrough'}
)
```

set_config(display="diagram")

When to use pipeline vs make_pipeline?

With Pipeline:
- Step/tranformer names are explicit,
- Ie. the name wotn't change if you change estimator/transformer used in a step,
  - e.g. if you replace LogisticRegression() with LinearSVC() you can still use clf__C.

With make_pipeline:
- shorter in use
- names are auto-generated using a straightforward rule (lowercase name of an estimator).
- Cons: if you change the function, names inside will change, an some other functions may stop working in the pipeline,

Thus:
- Use make_pipeline for quick experiments and
- Use Pipeline for more stable code/larger project
- not a big deal to use make_pipeline/Pipeline interchangeably

make_pipeline()

Here used to create multistep transformer
- Give names to steps automatically,
- Useful for pre-processing steps

FunctionTransformer()

Provides skilearn transformer API to custom functions
- ie. fit(), trasnform(), etc ...
- Allows using custom function with pipeline() & make_pieline to build more complex, mutistep tranformers/pieplines

HOW TO USE IT?

ColumnTransformer()

for: MIXED TRANFORMER TYPES

Allows applying different transformers to different columns in one dataset
- List with tranformer + column names (in a list)
- Option to leave some columns without chnages,
  - See, **"passthrough"**
- Option to remove, other columns,
  - See, **"drop"**

HOW TO BUILD CUSTOM TRANFORMERS

HOW TO USE IT?

```
# Example 1.  build a transformer with a log transformation
>>> import numpy as np
>>> from sklearn.preprocessing import FunctionTransformer
>>> transformer = FunctionTransformer(np.log1p, validate=True)
>>> transformer.fit_transform(X)
    # validate=True; check if the imput is an array

# ensure that inverse_tranform is possible
>>> enc = transformer.fit(X, check_inverse=True)
    # if not, returns a warning
    # Use fit, before transform, to apply it.

# Example 2. provide stats for twitter posts, eg. post length, and sentence nr.
>>> def text_stats(posts):
...         return [{'length': len(text),
...                  'num_sentences': text.count('.')}
...                  for text in posts]
>>> text_stats_transformer = FunctionTransformer(text_stats)

# Example 3. just add 1 to each value in a transformed columns
>>> def cust_func(x):
....        return x +1
```

SMALL PIPELINE WITH ESTIMATOR

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score

# load the data
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target

# split to train/test
>>> X_train, X_test, y_train, y_test = train_test_split(
                                X, y, train_size=0.7, random_state=0)

# scale input data
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)

# create classifier
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)

# predict & test
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Data Preprocessing

APPLYING MIXED TYPE TRANSFORMERS

SimpleAI.ch

THE SIMPLEST PIEPLINE WITH MIXED TRANFORMER TYPES

```
# Author: Pedro Morales <part.morales@gmail.com>
#
# License: BSD 3 clause

from __future__ import print_function

import pandas as pd
import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV

np.random.seed(0)

# Read data from Titanic dataset.
titanic_url = ('https://raw.githubusercontent.com/amueller/'
               'scipy-2017-sklearn/091d371/notebooks/datasets/titanic3.csv')
data = pd.read_csv(titanic_url)

# We will train our classifier with the following features:
# Numeric Features:
# - age: float.
# - fare: float.
# Categorical Features:
# - embarked: categories encoded as strings {'C', 'S', 'Q'}.
# - sex: categories encoded as strings {'female', 'male'}.
# - pclass: ordinal integers {1, 2, 3}.

# We create the preprocessing pipelines for both numeric and categorical data.
numeric_features = ['age', 'fare']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['embarked', 'sex', 'pclass']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(solver='lbfgs'))])

X = data.drop('survived', axis=1)
y = data['survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

clf.fit(X_train, y_train)
print("model score: %.3f" % clf.score(X_test, y_test))
```

1. **Define names of eg. categorical & numeric features**
   - *List[] with feature names*
2. **Define tranformation steps for each feature type**
   - *Pipepline(steps=[])*
   - *make_pipeline()*
   - *FunctionTransformer()*
3. **Create Preprocessor With mini-pipelines for different feature types**
   - *ColumnTranformer()*
4. **Add preprocessor to larger pipeline, that also contains the model to fit**
   - *Pipepline(steps=[("prep", preprocessor()), ("clasif", ml_alg_name())])*
5. **Prepare data (train/test split) Fit the model & Evaluate it**
   - *X,y = input_data*
   - *train_test_split(x,y , test_size=0.3)*
   - *Pipeline.fit(x_train, y_train)*
   - *Pipeline.score(X_test, y_Test)*
   - *Piepline.predict(X_test, y_Test)*

Similar example, provided by skilearn with some options for automated interactions with input data in pandas dataframe

Column Transformer with Mixed Types

This example illustrates how to apply different preprocessing and feature extraction pipelines to different subsets of features, using `ColumnTransformer`. This is particularly handy for the case of datasets that contain heterogeneous data types, since we may want to scale the numeric features and one-hot encode the categorical ones.

In this example, the numeric data is standard-scaled after mean-imputation, while the categorical data is one-hot encoded after imputing missing values with a new category (`'missing'`).

In addition, we show two different ways to dispatch the columns to the particular pre-processor: by column names and by column data types.

Finally, the preprocessing pipeline is integrated in a full prediction pipeline using `Pipeline`, together with a simple classification model.

```
# Author: Pedro Morales <part.morales@gmail.com>
#
# License: BSD 3 clause

import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.datasets import fetch_openml
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV

np.random.seed(0)

# Load data from https://www.openml.org/d/40945
X, y = fetch_openml("titanic", version=1, as_frame=True, return_X_y=True)

# Alternatively X and y can be obtained directly from the frame attributes:
# X = titanic.frame.drop('survived', axis=1)
# y = titanic.frame['survived']
```

Use `ColumnTransformer` by selecting column by names

We will train our classifier with the following features:

Numeric Features:

- age: float;
- fare: float.

Categorical Features:

- embarked: categories encoded as strings {'C', 'S', 'Q'};
- sex: categories encoded as strings {'female', 'male'};
- pclass: ordinal integers {1, 2, 3}.

We create the preprocessing pipelines for both numeric and categorical data. Note that `pclass` could either be treated as a categorical or numeric feature.

```
numeric_features = ['age', 'fare']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['embarked', 'sex', 'pclass']
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression())])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)

clf.fit(X_train, y_train)
print("model score: %.2f" % clf.score(X_test, y_test))
```

Out: `model score: 0.790`

HOW TO USE DIFFERENT TRANSFORMERS FOR DIFFERENT COLUMNS

Defining dtypes for mixed tranformers automatically

Select columns in pd.DataFrame by their dtypes using skilearn selector

df.info()

Used to fiund out how the columns will be categorized with the selecto

make_column_selector()

Can select columns based on

(i) Pandas data frame dtypes

(ii) or the columns name with a regex.

CAUTION: When using multiple selection criteria, **all** criteria must match for a column to be selected.

add i. supported Pandas df dtypes:

- ***numeric:*** *use "numeric"* , np.number, 'number'
- **categorical:** use "category"
- **object:** use object - this included str, and all other columns encoded as object (may be of any dtype)
- **datetimes:** use np.datetime64, 'datetime' or 'datetime64'
- **timedeltas:** use np.timedelta64, 'timedelta' or 'timedelta64'
- **datetimetz:** use 'datetimetz', or 'datetime64[ns, tz]'

```
>>> from sklearn.compose
...     import make_column_selector
...     as selector
>>> preprocessor = ColumnTransformer([
....     (num, numeric_transformer,
....     Selector(dtype_include="Category"),
```

MAPRAMETERS:

- **Pattern**
- **dtype_include**
- **dtype_exclude**

Example with selecting categorical and numerical dtypes

Use `ColumnTransformer` by selecting column by data types

When dealing with a cleaned dataset, the preprocessing can be automatic by using the data types of the column to decide whether to treat a column as a numerical or categorical feature. `sklearn.compose.make_column_selector` gives this possibility. First, let's only select a subset of columns to simplify our example.

```
subset_feature = ['embarked', 'sex', 'pclass', 'age', 'fare']
X_train, X_test = X_train[subset_feature], X_test[subset_feature]
```

Then, we introspect the information regarding each column data type.

```
X_train.info()
```

Out:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1047 entries, 1118 to 684
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   embarked  1045 non-null   category
 1   sex       1047 non-null   category
 2   pclass    1047 non-null   float64
 3   age       841 non-null    float64
 4   fare      1046 non-null   float64
dtypes: category(2), float64(3)
memory usage: 35.0 KB
```

We can observe that the `embarked` and `sex` columns were tagged as `category` columns when loading the data with `fetch_openml`. Therefore, we can use this information to dispatch the categorical columns to the `categorical_transformer` and the remaining columns to the `numerical_transformer`.

Note: In practice, you will have to handle yourself the column data type. If you want some columns to be considered as `category`, you will have to convert them into categorical columns. If you are using pandas, you can refer to their documentation regarding Categorical data.

```
from sklearn.compose import make_column_selector as selector

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, selector(dtype_exclude="category")),
    ('cat', categorical_transformer, selector(dtype_include="category"))
])
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression())])


clf.fit(X_train, y_train)
print("model score: %.3f" % clf.score(X_test, y_test))
```

Out: `model score: 0.794`

The resulting score is not exactly the same as the one from the previous pipeline because the dtype-based selector treats the `pclass` column as a numeric feature instead of a categorical feature as previously:

```
selector(dtype_exclude="category")(X_train)
```

Out: `['pclass', 'age', 'fare']`

```
selector(dtype_include="category")(X_train)
```

Out: `['embarked', 'sex']`

sparse matrices vs numpy arrays

https://stackoverflow.com/questions/36969886/using-a-sparse-matrix-versus-numpy-array

HOW TO VISUALIZE THE PIPELINE

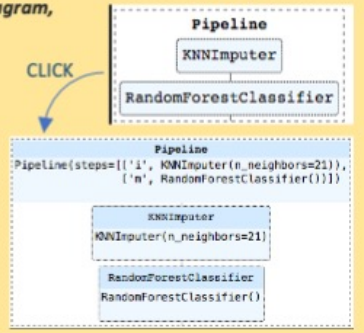
set_configs()

Use function that set global scikit-learn configuration And ask for displaying nice diagram, instead of text

CAUTION: it changes all global settings!

```
>>> from sklearn import set_config
>>> set_config(display="diagram")
    # "text" is an alternative, and
      normally used.
>>> pipeline
```

CLICK



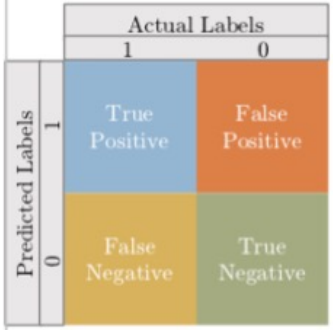
DEALING WITH BIASED DATASETS

Classifier that always predicts label blue yields prediction accuracy of 90%

Accuracy doesn't always give the correct insight about your trained model

Accuracy:	%age correct prediction	Correct prediction over total predictions	One value for entire network
Precision:	Exactness of model	From the detected cats, how many were actually cats	Each class/label has a value
Recall:	Completeness of model	Correctly detected cats over total cats	Each class/label has a value
F1 Score:	Combines Precision/Recall	Harmonic mean of Precision and Recall	Each class/label has a value

Performance metrics associated with Class 1



(Is your prediction correct?) (What did you predict)

True Negative

(Your prediction is correct) (You predicted 0)

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1 score} = 2x \frac{(\text{Prec x Rec})}{(\text{Prec} + \text{Rec})}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{False +ve rate} = \frac{FP}{TN + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{Recall, Sensitivity} = \text{True +ve rate} = \frac{TP}{TP + FN}$$

Possible solutions

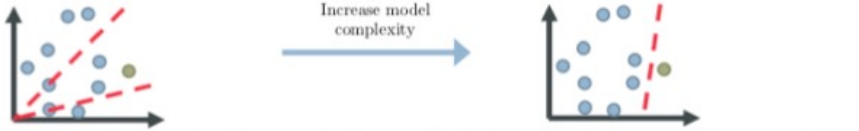
1. **Data Replication:** Replicate the available data until the number of samples are comparable

2. **Synthetic Data:** Images: Rotate, dilate, crop, add noise to existing input images and create new data

3. **Modified Loss:** Modify the loss to reflect greater error when misclassifying smaller sample set

$$loss = a * loss_{green} + b * loss_{blue} \qquad a > b$$

4. **Change the algorithm:** Increase the model/algorithm complexity so that the two classes are perfectly separable (Con: Overfitting)



No straight line (y=ax) passing through origin can perfectly separate data. **Best solution:** line y=0, predict all labels blue

Straight line (y=ax+**b**) can perfectly separate data. Green class will no longer be predicted as blue

Source: https://www.cheatsheets.aqeel-anwar.com

ADVANCED SKLEARN

by Pawel Rosikiewicz

Building Pipelines with Skilearn

www.SimpleAI.ch

BASIC PIPELINE

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
```

Load skilearn example dataset

```
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
```

Prepare the data

```
# split to train/test
>>> X_train, X_test, y_train, y_test = train_test_split(
                X, y, train_size=0.7, random_state=0)
# scale input data
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
```

Create classifier & train it

```
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5
>>> knn.fit(X_train, y_train)
```

Predict test values, & measure error

```
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_) pred
```

Organize multiple steps in a pipeline

CREATE PIPELINE WITH SKLEARN

Pipeline()

encapsulate multiple steps with pipeline

```
# each step requires a name, and a function
>>> pipe = Pipeline([
            ('scaler', preprocessing.StandardScaler()),
            ('knn',neighbors.KNeighborsClassifier(n_neighbors=5))
           ])

# Multiple preprocessing steps can be added
>>> pipe = Pipeline([
            ('transform_1', preprocessing.function_1()),
            ('transform_1', preprocessing.function_1()),
            ('transform_1', preprocessing.function_1()),
             ...,
            ('knn',neighbors.KNeighborsClassifier(n_neighbors=5))
           ])
```

API

```
# You can use standard skileanr API with the pipeline
>>> pipe.fit(X,y)
>>> pipe.sore(X,y)
>>> pipe.predict(X,y)
>>> MAE(y, pipe.predict(X,y))
```

Caution Accessing some outputs of each step may be more complicated

Control over steps & parameters

```
# Check the pipeline
>>> pipe.named_steps                    # basic text info
>>> pipe.get_params()                   # to see parameters at each step
>>> set_config(display="diagram"); pipe.  # beatiful visualizations (HTML)

# disable some steps
>>> from= Pipeline([
            ('scaler', None)
            ('knn',neighbors.KNeighborsClassifier(n_neighbors=5))
           ])

# set parameters for individual steps
"' <step_name>__<parameter_name>""
>>> pipe(knn__n_neighbors=<value>)
```

Replace them, with one custom transformer

ADD MORE COMPLICATED STEPS INTO THE PIPE

make_pipeline()

- Give names to steps automatically,
- Useful for pre-processing steps

FunctionTransformer()

Provides skilearn transformer API to custom functions

- ie. fit(), trasnform(), etc ...
- Allows using custom function with pipeline() & make_pieline to build more complex, mutistep tranformers/pieplines

```
# create custom transformer
'' unlike pipeline, make_pipepline
  generates names for steps automatically
  ->> lowercase name of an estimator
  otherwise, these two fucntions work in the same way
>>> import numpy as np
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import FunctionTransformer
>>> from sklearn.preprocessing import StandardScaler,

>>> log_scale_transformer = make_pipeline(
...                         FunctionTransformer(np.log, validate=False),
...                         StandardScaler()
...                         )
```

USE DIFFERENT TRANSFORMERS FOR DIFFERENT COLUMNS

ColumnTransformer()

Allows applying different transformers to different columns in one dataset

- List with transformer + column names (in a list)
- Option to leave some columns without changes,
  - See, "passthrough"
- Option to remove, other columns,
  - See, "drop"

```
# Create fdata preprocessor with ColumnTransformer()
"' ColumnTransformer takes a list with instrucitons for each column/col. Group.
  for each of them you must provide the folloiwing
        (i)  unique preprocessor name
        (ii). Transformer function, "passthrough" str, to not make any modiff's
              - if more then one function, you need to create it separately,
                using pieline or make pipline,
                like log_scale_transformer in this example.
        (iii) column names in input data, that will be transformed  (LIST)
"'
>>> from sklearn.compose import ColumnTransformer
>>> preprocessor = ColumnTransformer(
        transformers=[
            ("passthrough_numeric", "passthrough", ["column_1"]),
            ("binned_numeric", KBinsDiscretizer(n_bins=10), [" column_2", " col..."]),
             ("log_scaled_numeric", log_scale_transformer, [" column_4"]),
             ("onehot_categorical", OneHotEncoder(), [" column_5", " column_5"]),
        ],
        remainder="drop",     # TWO OPTION {'drop', 'passthrough'}
       )
```

TUNE HYPERPARAMETERS

Used for grid search

```
# test different k values
>>>  results = []
>>> for k in list(range(1,10)):
...                     # set k in the pipeline
...                     pipe(knn__n_neighbors=i)
...                     pipe.fit(X_tr, y_tr)
..
...                     # collect the results
                        gs_results.append(
                                'k':k,
                                test_mae: MAE(y_te, pipe.predict(X_te))
```

Report results

```
# convert the results to pd.dataframe & plot
>>> gs = pd.DataFrame(gs_results)
>>> plt.plot(gs['k'], gs['test_mae'])
>>> plt.xlabel,   plt.legend(),   plt.show() ....
```

df.info()

Used to find out how the columns will be categorized with the selector

supported Pandas df dtypes:

- *numeric:* use "numeric" , np.number, 'number'
- categorical: use "category"
- object: use object - this included str, and all other columns encoded as object (may be of any dtype)
- datetimes: use np.datetime64, 'datetime' or 'datetime64'
- and other .... See my notes on transformers,

make_column_selector()

Can select columns inpandas df based on

(i) Pandas data frame dtypes
(ii) columns name with a regex.

CAUTION: When using multiple selection criteria, **all** criteria must match for a column to be selected.

If different transformers are set based on dtype in the data, you may set them automatically, With skilearn function

```
>>> from sklearn.compose
...     import make_column_selector  as selector
>>> preprocessor = ColumnTransformer([
....      (num, numeric_transformer,
....      Selector(dtype_include="Category"),
```

- *Pattern*
- *dtype_include*
- *dtype_exclude*

ParameterGrid()

Next Slide

by Pawel Rosikiewicz

Building Pipelines with Skilearn

www.SimpleAI.ch

BASIC PIPELIN WITH CV & GRIDSEARCH

Data preparation

train_test_split()

You may use more advanced options with k-fold

CV allows repeating this part on subset of the data to find optimal hyperparameters

Data transformers

imputers

Model

ParameterGrid()

May also be used for some transformers and imputers

Estimator function

In many cases estimator allows selecting different error functions

Evaluation

est.predict()

sklearn.metrics()

Baseline (eg: R^2)

What are hyperparameters?

Def: a type of model parameters that are not directly learnt within estimators, from the data, Eg: C, kernel and gamma for Support Vector Classifier, alpha for Lasso, etc. You select them, based on model performance, and domain knowledge

 # to find the names and current values for all parameters:

```
>>> skleanr_estimator.get_params()
```

- OPTION 1 - TUNING HYPARAMETERS WITH GRIDSEARCH

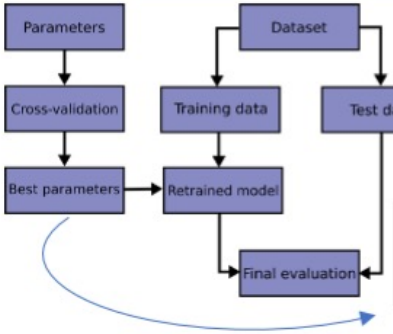
➢ ParameterGrid()

Function used to iterate over parameter value combinations

- Applies Python built-in function iter.
- The order of the generated parameter combinations deterministic.
- Can be accessed as any list or iterator
- ALL OTHER ELEMENTS OF THE PIELINE MUST BE ADDED

```
>>> from sklearn.model_selection import ParameterGrid

# option 1. takes dict with param name & their values
"""returns all combinations of these params""
>>> param_grid = {
        'a': [1, 2],
        'b': [True, False] } # list with each parameter
# may be accesses with for loop, or turn into the list
>>> list(ParameterGrid(param_grid)) == (
...   [{'a': 1, 'b': True}, {'a': 1, 'b': False},
...    {'a': 2, 'b': True}, {'a': 2, 'b': False}])

# option 2. may take several lists of dict with param. Names:values
"""it will return grid with comnbiation of all param values, in each
dictionary, but not between them !"""
>>> grid = [
        {'kernel': ['linear']},
        {'kernel': ['rbf'], 'gamma': [1, 10]
        }
# returns the following:
        list(ParameterGrid(grid)) ==
                [{'kernel': 'linear'},
...              {'kernel': 'rbf', 'gamma': 1},
...              {'kernel': 'rbf', 'gamma': 10}]
```

- OPTION 2 - TUNING HYPARAMETERS WITH CROSS_VALIDATION



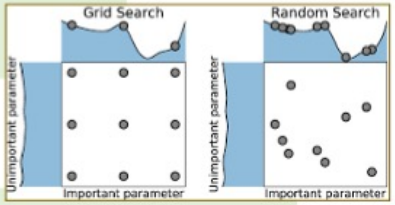

```
# to get the results in pd df
df = pd.DataFrame(.
      est.cv_results_)
```

	iter	n_resources	mean_test_score	params
0	0	125	0.983667	{'criterion': 'entropy', 'max_ 'max_features': 9, 'min_sa

BRUTE FORCE APPROACH

➢ GridSearchCV()

- **Brut Force Approach = Exhaustive Grid Search for provided parameter ranges, or values, specified with** with ParameterGrid() within that function
- Can return multiple scoring methods
- All combinations of parameters will be tested on entire provided dataset



➢ RandomizedSearchCV()

- **Parameters:**
  - **Hyperparameter parameters sampling** - done with a dictionary, similar to specifying parameters for GridSearchCV. Subsequently, these are randomly sampled from that sampe or distribution (sampled uniformly)

```
{'C': scipy.stats.expon(scale=100),
 'gamma': scipy.stats.expon(scale=.1),
 'kernel': ['rbf'],
 'class_weight':['balanced', None]}
```

  - **N_iter** – additional, parameter controlling how many times the algorithm should samples from hypeparam. Space
- ***COMMENT:***
  - **Use Scipy.stats module -** contains many useful distributions for sampling parameters, eg: expon, gamma, uniform or randint
  - **For continuous parameters, such as C** – you must specify a continuous distribution to take full advantage of the randomization. This way, increasing n_iter will always lead to a finer search.
  - For log-uniform use:

```
>>> loguniform(1, 100)  # returns[1, 10, 100]
Or
>>> np.logspace(0, 2, num=1000)
# example:
from sklearn.utils.fixes import loguniform
{'C': loguniform(1e0, 1e3), etc...
```

SUCCESSIVE HALVING APPROACH

- **Iterative approach**
- First, all candidates (the parameter combinations) are evaluated with a small amount of resources. Then, only a subset of param. Comb's are selected to next iteration.
- Each iteration is allocated an increasing amount of resources per candidate, eg. sample number.
- Typical Iteration parameters:
  - Sample number
    - Factor; >1, Default values ==2; controls the rate at which the resources grow, and the rate at which the number of candidates decreases
    - resource & min_resources
    - Eg: min_resources=10 and factor=2 params will give the following iterations in respect to sampel nr. [10, 20, 40, 80, 160, 320, 640]
  - More specific params – eg: n_estimators in a random forest
- **Best candidates are selected based on:**
  - param combinations, that have consistently ranked among the top-scoring candidates across all iterations.
  - only a subset of candidates 'survive' until the last iteration

➢ HalvingGridSearchCV()

➢ HalvingRandomizedSearchCV()

These two functions are sklearn alternatives to brute force CV functions (eg RandomizedCV) , that allow implementing Halving methods. *However,* These estimators are still **experimental**: their predictions and their API might change without any deprecation cycle (from sklearn)

YOPU MAY USE MANY SKLEARN FUNCTION WITH BUILD IN CV FUNCTIONALITY

`linear_model.ElasticNetCV`(*[, l1_ratio, ...])	Elastic Net model with iterative fitting along a regularization path.
`linear_model.LarsCV`(*[, fit_intercept, ...])	Cross-validated Least Angle Regression model.
`linear_model.LassoCV`(*[, eps, n_alphas, ...])	Lasso linear model with iterative fitting along a regularization path.
`linear_model.LassoLarsCV`(*[, fit_intercept, ...])	Cross-validated Lasso, using the LARS algorithm.
`linear_model.LogisticRegressionCV`(* [, Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.
`linear_model.MultiTaskElasticNetCV`(*[, ...])	Multi-task L1/L2 ElasticNet with built-in cross-validation.
`linear_model.MultiTaskLassoCV`(*[, eps, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer.
`linear_model.OrthogonalMatchingPursuitCV`(*)	Cross-validated Orthogonal Matching Pursuit model (OMP).
`linear_model.RidgeCV`([alphas, ...])	Ridge regression with built-in cross-validation.
`linear_model.RidgeClassifierCV`([alphas, ...])	Ridge classifier with built-in cross-validation.

TUNING HYPERPARAMETERS

GridSearch

Using for loops

```
# Example: test different k values
>>> Gs_results = []
>>> for k in list(range(1,10)):
...                     # set k in the pipeline
...                     pipe(knn__n_neighbors=i)
...                     pipe.fit(X_tr, y_tr)
..                      # collect the results
...                     gs_results.append(
...                                 'k':k,
...                                 test_mae: MAE(
                                            y_te, pipe.predict(X_te))
# convert the results to pd.dataframe & plot them
>>> gs = pd.DataFrame(gs_results)
>>> plt.plot(gs['k'], gs['test_mae'])
>>> plt.xlabel,   plt.legend(),   plt.show() ….
```

ParameterGrid()

Used to iterate over parameter value combinations

- Applies Python built-in function iter.
- The order of the generated parameter combinations is deterministic.
- Can be accessed as any list or iterator

```
>>> from sklearn.model_selection import ParameterGrid

# option 1. takes dict with param name & their values
"""returns all combinations of these params"""
>>> param_grid = {
            'a': [1, 2],
            'b': [True, False] } # list with each parameter
# may be accesses with for loop, or turn into the list
>>> list(ParameterGrid(param_grid)) == (
...    [{'a': 1, 'b': True}, {'a': 1, 'b': False},
...     {'a': 2, 'b': True}, {'a': 2, 'b': False}])

# option 2. may take several lists of dict with param.
Names:values
"""it will return grid with comnbiation of all param values,
in each dictionary, but not between them !"""
>>> grid = [
            {'kernel': ['linear']},
            {'kernel': ['rbf'], 'gamma': [1, 10]
          }
# returns the following:
          list(ParameterGrid(grid)) ==
                    [{'kernel': 'linear'},
...                  {'kernel': 'rbf', 'gamma': 1},
...                  {'kernel': 'rbf', 'gamma': 10}]
```

AUTOml

Auto-sklearn

https://automl.github.io/auto-sklearn/master/

FEATURES

- *Allows fast classification using many popular methods*
- *It can be extended with new classification, regression and feature pre-processing methods*

CONS

- *Limited knowledge on explored space*

EXAMPLE:

```
"""has fit, predict methods"""
>>> import autosklearn.classification
>>> cls = autosklearn.classification.AutoSklearnClassifier()
>>> cls.fit(X_train, y_train)
>>> predictions = cls.predict(X_test)
```

Visualize tested parameters

Pipeline Profiler Tool

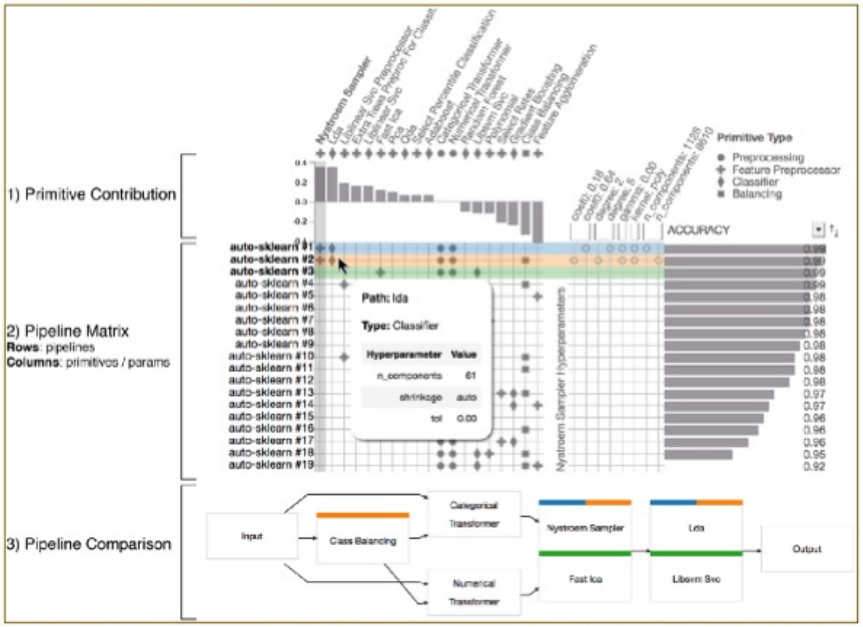
EXAMPLE

```
# classify digit dataset with automl skileanr package
>>> import sklearn.datasets
>>> import autosklearn.classification

>>> X, y = sklearn.datasets.load_digits(return_X_y=True)
>>> automl = autosklearn.classification.AutoSklearnClassifier()
>>> automl.fit(X, y, dataset_name='digits')

# Visualize results with pipeline profiler
""" it helps understanding what have happened"""
>>> import PipelineProfiler
>>> profiler_data = PipelineProfiler.import_autosklearn(automl)
>>> PipelineProfiler.plot_pipeline_matrix(profiler_data)
      # creates  the plot in the above
```

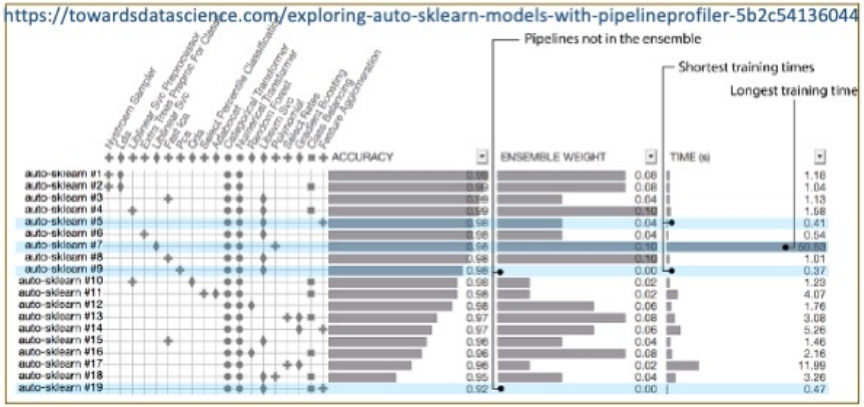
This is example of a new tool that can help with automl solutions

Using pipeline proficler we can we can see which pipeline elements:

- *were corelated with high accuracy scores*
- *contributed to long training time*
- *And many more*



https://towardsdatascience.com/exploring-auto-sklearn-models-with-pipelineprofiler-5b2c54136044



APPLYING CUSTOM TRANSFORMERS

EXAMPLE: APPLYING CUSTOM TRANFORMERS

```
>>> import numpy as np
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import FunctionTransformer
>>> from sklearn.preprocessing import OneHotEncoder, StandardScaler, KBinsDiscretizer
>>> from sklearn.compose import ColumnTransformer

# 1. DEFINE CUTOM TRANFROMERS
# create custom transformer
'' unlike pipeline, make_pipepline
  generates names for steps automatically
  ->> lowercase name of an estimator
  otherwise, these two fucntions work in the same way
'''
>>> log_scale_transformer = make_pipeline(
...                         FunctionTransformer(np.log, validate=False),
...                         StandardScaler()
...                         )

# 2. DEFINE PREPROCESSING FUNCTION WITH DIFFERENT  TRANFORMATIONS
      FOR DIFFERENT COLUMNS
# Create final preprocessor for the data, with ColumnTransformer()
''' ColumnTransformer takes a list with instrucitons for each column/col. Group.
   for each of them you must provide the folloiwing
            (i)  unique preprocessor name
            (ii). Transformer function, "passthrough" str, to not make any modifications
                        - if more then one function, you need to create it separately,
                          using pieline or make pipline,
                          like log_scale_transformer in this example.
            (iii) column names in input data, that will be transformed  (LIST)
'''
linear_model_preprocessor = ColumnTransformer(
   transformers=[

      ("passthrough_numeric", "passthrough", ["column_1"]),

      ("binned_numeric", KBinsDiscretizer(n_bins=10), [" column_2", " column_ 3"]),

      ("log_scaled_numeric", log_scale_transformer, [" column_4"]),

      ("onehot_categorical", OneHotEncoder(), [" column_5", " column_5", " column_6"]),
   ],
   remainder="drop",     # TWO OPTION {'drop', 'passthrough'}
)
```

set_config(display="diagram")

When to use pipeline vs make_pipeline?

With Pipeline:
- Step/tranformer names are explicit,
- Ie. the name wotn't change if you change estimator/transformer used in a step,
  - e.g. if you replace LogisticRegression() with LinearSVC() you can still use clf__C.

With make_pipeline:
- shorter in use
- names are auto-generated using a straightforward rule (lowercase name of an estimator).
- Cons: if you change the function, names inside will change, an some other functions may stop working in the pipeline,

Thus:
- Use make_pipeline for quick experiments and
- Use Pipeline for more stable code/larger project
- not a big deal to use make_pipeline/Pipeline interchangeably

make_pipeline()

Here used to create multistep transformer
- Give names to steps automatically,
- Useful for pre-processing steps

FunctionTransformer()

Provides skilearn transformer API to custom functions
- ie. fit(), trasnform(), etc ...
- Allows using custom function with pipeline() & make_pieline to build more complex, mutistep tranformers/pieplines

HOW TO USE IT?

ColumnTransformer()

for: MIXED TRANFORMER TYPES

Allows applying different transformers to different columns in one dataset
- List with tranformer + column names (in a list)
- Option to leave some columns without chnages,
  - See, **"passthrough"**
- Option to remove, other columns,
  - See, **"drop"**

HOW TO BUILD CUSTOM TRANFORMERS

HOW TO USE IT?

```
# Example 1.  build a transformer with a log transformation
>>> import numpy as np
>>> from sklearn.preprocessing import FunctionTransformer
>>> transformer = FunctionTransformer(np.log1p, validate=True)
>>> transformer.fit_transform(X)
    # validate=True; check if the imput is an array

# ensure that inverse_tranform is possible
>>> enc = transformer.fit(X, check_inverse=True)
    # if not, returns a warning
    # Use fit, before transform, to apply it.

# Example 2. provide stats for twitter posts, eg. post length, and sentence nr.
>>> def text_stats(posts):
...         return [{'length': len(text),
...                  'num_sentences': text.count('.')}
...                  for text in posts]
>>> text_stats_transformer = FunctionTransformer(text_stats)

# Example 3. just add 1 to each value in a transformed columns
>>> def cust_func(x):
....        return x +1
```

SMALL PIPELINE WITH ESTIMATOR

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score

# load the data
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target

# split to train/test
>>> X_train, X_test, y_train, y_test = train_test_split(
                                X, y, train_size=0.7, random_state=0)

# scale input data
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)

# create classifier
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)

# predict & test
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

APPLYING MIXED TYPE TRANSFORMERS

THE SIMPLEST PIEPLINE WITH MIXED TRANFORMER TYPES

```
# Author: Pedro Morales <part.morales@gmail.com>
#
# License: BSD 3 clause

from __future__ import print_function

import pandas as pd
import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV

np.random.seed(0)

# Read data from Titanic dataset.
titanic_url = ('https://raw.githubusercontent.com/amueller/'
               'scipy-2017-sklearn/091d371/notebooks/datasets/titanic3.csv')
data = pd.read_csv(titanic_url)

# We will train our classifier with the following features:
# Numeric Features:
# - age: float.
# - fare: float.
# Categorical Features:
# - embarked: categories encoded as strings {'C', 'S', 'Q'}.
# - sex: categories encoded as strings {'female', 'male'}.
# - pclass: ordinal integers {1, 2, 3}.

# We create the preprocessing pipelines for both numeric and categorical data.
numeric_features = ['age', 'fare']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['embarked', 'sex', 'pclass']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(solver='lbfgs'))])

X = data.drop('survived', axis=1)
y = data['survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

clf.fit(X_train, y_train)
print("model score: %.3f" % clf.score(X_test, y_test))
```

1. **Define names of eg. categorical & numeric features**
   - *List[] with feature names*
2. **Define tranformation steps for each feature type**
   - *Pipepline(steps=[])*
   - *make_pipeline()*
   - *FunctionTransformer()*
3. **Create Preprocessor With mini-pipelines for different feature types**
   - *ColumnTranformer()*
4. **Add preprocessor to larger pipeline, that also contains the model to fit**
   - *Pipepline(steps=[("prep", preprocessor()), ("clasif", ml_alg_name())])*
5. **Prepare data (train/test split) Fit the model & Evaluate it**
   - *X,y = input_data*
   - *train_test_split(x,y , test_size=0.3)*
   - *Pipeline.fit(x_train, y_train)*
   - *Pipeline.score(X_test, y_Test)*
   - *Piepline.predict(X_test, y_Test)*

Similar example, provided by skilearn with some options for automated interactions with input data in pandas dataframe

Column Transformer with Mixed Types

This example illustrates how to apply different preprocessing and feature extraction pipelines to different subsets of features, using `ColumnTransformer`. This is particularly handy for the case of datasets that contain heterogeneous data types, since we may want to scale the numeric features and one-hot encode the categorical ones.

In this example, the numeric data is standard-scaled after mean-imputation, while the categorical data is one-hot encoded after imputing missing values with a new category (`'missing'`).

In addition, we show two different ways to dispatch the columns to the particular pre-processor: by column names and by column data types.

Finally, the preprocessing pipeline is integrated in a full prediction pipeline using `Pipeline`, together with a simple classification model.

```
# Author: Pedro Morales <part.morales@gmail.com>
#
# License: BSD 3 clause

import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.datasets import fetch_openml
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV

np.random.seed(0)

# Load data from https://www.openml.org/d/40945
X, y = fetch_openml("titanic", version=1, as_frame=True, return_X_y=True)

# Alternatively X and y can be obtained directly from the frame attribute:
# X = titanic.frame.drop('survived', axis=1)
# y = titanic.frame['survived']
```

Use `ColumnTransformer` by selecting column by names

We will train our classifier with the following features:

Numeric Features:

- age: float;
- fare: float.

Categorical Features:

- embarked: categories encoded as strings {'C', 'S', 'Q'};
- sex: categories encoded as strings {'female', 'male'};
- pclass: ordinal integers {1, 2, 3}.

We create the preprocessing pipelines for both numeric and categorical data. Note that `pclass` could either be treated as a categorical or numeric feature.

```
numeric_features = ['age', 'fare']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['embarked', 'sex', 'pclass']
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression())])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)

clf.fit(X_train, y_train)
print("model score: %.2f" % clf.score(X_test, y_test))
```

Out: `model score: 0.790`

HOW TO USE DIFFERENT TRANSFORMERS FOR DIFFERENT COLUMNS

Defining dtypes for mixed tranformers automatically

Select columns in pd.DataFrame by their dtypes using skilearn selector

df.info()

Used to fiund out how the columns will be categorized with the selecto

make_column_selector()

Can select columns based on

(i) Pandas data frame dtypes

(ii) or the columns name with a regex.

CAUTION: When using multiple selection criteria, **all** criteria must match for a column to be selected.

add i. supported Pandas df dtypes:

- ***numeric:*** *use "numeric"* , np.number, 'number'
- **categorical:** use "category"
- **object:** use object - this included str, and all other columns encoded as object (may be of any dtype)
- **datetimes:** use np.datetime64, 'datetime' or 'datetime64'
- **timedeltas:** use np.timedelta64, 'timedelta' or 'timedelta64'
- **datetimetz:** use 'datetimetz', or 'datetime64[ns, tz]'

```
>>> from sklearn.compose
...     import make_column_selector
...     as selector
>>> preprocessor = ColumnTransformer([
....     (num, numeric_transformer,
....     Selector(dtype_include="Category"),
```

MAPRAMETERS:

- **Pattern**
- **dtype_include**
- **dtype_exclude**

Example with selecting categorical and numerical dtypes

Use `ColumnTransformer` by selecting column by data types

When dealing with a cleaned dataset, the preprocessing can be automatic by using the data types of the column to decide whether to treat a column as a numerical or categorical feature. `sklearn.compose.make_column_selector` gives this possibility. First, let's only select a subset of columns to simplify our example.

```
subset_feature = ['embarked', 'sex', 'pclass', 'age', 'fare']
X_train, X_test = X_train[subset_feature], X_test[subset_feature]
```

Then, we introspect the information regarding each column data type.

```
X_train.info()
```

Out:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1047 entries, 1118 to 684
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   embarked  1045 non-null   category
 1   sex       1047 non-null   category
 2   pclass    1047 non-null   float64
 3   age       841 non-null    float64
 4   fare      1046 non-null   float64
dtypes: category(2), float64(3)
memory usage: 35.0 KB
```

We can observe that the `embarked` and `sex` columns were tagged as `category` columns when loading the data with `fetch_openml`. Therefore, we can use this information to dispatch the categorical columns to the `categorical_transformer` and the remaining columns to the `numerical_transformer`.

Note: In practice, you will have to handle yourself the column data type. If you want some columns to be considered as `category`, you will have to convert them into categorical columns. If you are using pandas, you can refer to their documentation regarding Categorical data.

```
from sklearn.compose import make_column_selector as selector

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, selector(dtype_exclude="category")),
    ('cat', categorical_transformer, selector(dtype_include="category"))
])
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression())])


clf.fit(X_train, y_train)
print("model score: %.3f" % clf.score(X_test, y_test))
```

Out: `model score: 0.794`

The resulting score is not exactly the same as the one from the previous pipeline because the dtype-based selector treats the `pclass` column as a numeric feature instead of a categorical feature as previously:

```
selector(dtype_exclude="category")(X_train)
```

Out: `['pclass', 'age', 'fare']`

```
selector(dtype_include="category")(X_train)
```

Out: `['embarked', 'sex']`

sparse matrices vs numpy arrays

https://stackoverflow.com/questions/36969886/using-a-sparse-matrix-versus-numpy-array

HOW TO VISUALIZE THE PIPELINE

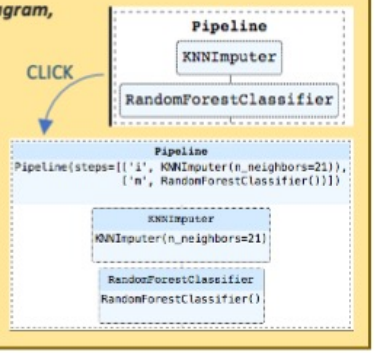
set_configs()

Use function that set global scikit-learn configuration And ask for displaying nice diagram, instead of text

CAUTION: it changes all global settings!

```
>>> from sklearn import set_config
>>> set_config(display="diagram")
    # "text" is an alternative, and
      normally used.
>>> pipeline
```

CLICK



FEATURE SELECTION